Man machine interface for real time power system simulation

Ng, F.

Award date:
1992

Awarding institution:
University of Bath

MAN MACHINE INTERFACE FOR REAL TIME POWER SYSTEM SIMULATION

Submitted by F. Ng, B.Sc.(Hons)
for the degree of
Doctor of Philosophy
of the University of Bath
1992

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Contents

Summary

This thesis describes research into the design issues of implementing a man machine interface, MMI, for a power system simulator which is used to simulate the electro-mechanical transient behaviour of the British National Grid transmission system in real time. The computer system used is a locally designed multiprocessor transputer system, running the Helios operating system. The MMI follows the modern practice of user interface design by implementing a graphical user interface and the X Window System provides the main interface building tools.

The MMI has been successfully shown to be user friendly for the novice user while at the same time providing powerful features for the more experienced user. In addition, it opens up scope for future research into the new direction of user interface design for real time power system simulators.

Acknowledgements

The author is indebted to his academic supervisor, Mr A.R. Daniels for his help and guidance throughout this work. He would like to thank Professor J.F. Eastham, Head of School for provision of facilities at the School of Electrical Engineering and also the National Grid Company (NGC) for the use of their facilities.

Financial support from the Science and Engineering Research Council and the NGC is gratefully acknowledged.

The author would like to express his thanks to his colleagues at the University and staff at the NGC, in particular Dr T. Berry, Dr R.W. Dunn and Mr V.S. Gott for their help and advice throughout this project.

Finally, the author would also like to thank his brother Mr. K. Ng for drawing some of the figures in the thesis.

This thesis was produced using LaTeX running under the Unix operating system.

Glossary

Dialogue box
Arbitrary input data values cannot be supplied via a menu dialog. A window can be popped up on the screen (called a popup window) where a number of pre-determined questions will be presented for the user.

Drop Down Menu
This appears automatically when a user moves along the menu bar. A subsidiary menu block will "drop down" under each heading.

GEM : Graphical Environment Manager
This is a Digital Research's user interface, which was one of the first windowing systems. Users interact with the system via graphical objects displayed on the screen with either a keyboard or a mouse as input devices.

GUI: Graphical User Interface
This is the general name for an interactive graphical environment in which the user interacts with the system via graphical objects on the display.

Hypertext
This is used to describe a specialised type of database. The principle is that keywords exist within the text of the information displayed on-screen, and selecting those keywords will produce a link to a subsequent screen, which may be the next part of a program, or may be a help screen that provides more detailed information on the selected keyword. This approach is particularly popular in a graphic-based operating system, where a mouse can be used to highlight and select the hypertext.

Icon
In computing terms, an icon is a graphical representation of a physical item, usually either a file or a peripheral, so that, for example, a small picture of a disk drive is displayed on-screen to indicate the actual presence of a disk drive.

Menu Bar
This is normally displayed in a window at the top of the screen . Permanently available functions are classified into groups and the group headings are displayed as the options of a menu bar.

PopUp Menu
This is a window which appears usually in the middle of the display, requesting the user to select from a menu block. The user selects by scrolling through the options and the window disappears when the selection is made.

Pull Down Menu
This appears only if the user actually selects an entry in the menu bar by clicking on it with a pointer. A subsidiary menu block will "drop down" under each heading for further selection.

Real Time
This is used to describe a computer system whose response is fast enough to satisfy human users. In simulation terms, the simulation model equations must be solved by the computer within a given model timestep. Berry [1] has further defined the terms Hard Real Time and Soft Real Time: A Hard Real Time system is one which has "the solution time equals to the model time step under all conditions with output synchronized with the real (clock) time". A Soft Real Time system is one which has "the soultion time generally be less than the model timestep but occassionally may be greater".

Tagging
This is an item used to describe the process of marking a series of items for later processing. This might involve marking a group of files to be deleted from a disk drive, or perhaps a set of elements of a picture that are to be printed in a particular colour.

UIMSs: User Interface Management Systems.
They are software packages that support the implementation of user interfaces which are both portable and consistent. Strict separation of the interface from the application is encouraged, so that the user and the application do not communicate directly, but only via a UIMS.

WIMP
W - present information to users via multiple Windows on the display.
I - representing pictorially as Icons the data objects with which the system is concerned.
M - using a Mouse to select graphical objects.
P - menus which popup automatically on the screen or which a user can pull down from a menu bar.

Window
This is a specified (usually rectangular) area of the physical screen through which a user views a particular aspect of the interaction with a particular task.

Window Manager
This is a X window client that provides a screen layout policy and allocates resources for all the clients running concurrently on the same server. Re-sizing and decorating windows are its common responsibilities. Effectively, it determines the "look and feel" of an application.

WYSIWYG What You See Is What You Got.
This describes interfaces in which the actual effect of any action is reflected upon the display immediately. This effectively provides a natural status message confirming that the action has been carried out.

X Athena Widget Set
This is a set of programming objects built upon the Xtk ToolKit. It provides an even higher level of abstraction than Xtk ToolKit and allows programming in a limited object-oriented style.

Xtk ToolKit
This is a set of interface components and control mechanism built upon the basic libraries of the X Window System. It provides a policy free, higher abstraction programming utility for interface programmer to build a graphical user interface.

X Window System
This was developed at MIT as a distributed, network transparent, device independent, multitasking windowing and graphic system. It is based on the client/server model in which an application program is run independently as a client while the system is run as a server, servicing the client via protocol calls. Multiple applications can be displayed on the same screen and each one can use many windows. Overlapping and hidden windows, text with soft fonts, and two-dimensional graphic drawings are supported.

List of Principal Symbols

CPU	Central Processing Unit.
DMA	Direct Memory Access.
DRAM	Dynamic Random Access memory
EPROM	Erasable Programmable ROM.
I/O	Input/Output
MHz	Frequecy in million cycles per second.
MMI	The research project: Man Machine Interface.
N-way crossbar	A large bank of switches that can be controlled to allow any combination of connections of N inputs and N outputs.
RISC	Reduced Instruction Set Computer.
ROM	Read Only Memory
Simulator	The latest version of the Real Time Power System Simulator developed at Bath University.
SRAM	Static Random Access memory
UART	Universal Asynchronous Receive and Transmit Device.
VLSI	Very Large Scale Integrated Circuit.
VSC	Video and System Controller.

Chapter 1

Introduction

1.1 Power System Simulation

A system for the generation and distribution of electrical power is one of the most important and complex utilities of modern life. With an ever increasing demand from the public and industry, the system is always pushed to operate at a very high standard. With the recent public concern for the environment and the privatization of the Central Electricity Generating Board and the twelve regional electricity boards, the British power system is expected to perform even better. This places a considerable demand on the planning of new power plants and the training of power system operators.

The British National Grid is a very complex network, which is made up of more than eighty generators, four hundred busbars and one thousand two hundred lines. There exists the need to provide some aids to the power system engineers in analyzing such a transmission system. In particular, the transient stability study of a power system subjected to severe disturbances such as network faults and outages are especially important in accessing the stability performance of the

system. The requirement can best be satisfied by a high performance real time power system simulator which is capable of producing fast and accurate results of the power system under study.

1.2 Bath University Real Time Power System Simulator

Bath University has been active in the field of the development of simulators for studying the electro-mechanical transient behaviour of the British power system since the early 1980s. Dale [2] simulated a four machine power system which laid the foundation for Berry [3], who further developed the simulator and expanded it to run a twenty machine study. Both simulators were producing results in real time. Over the years, as the power system to be studied became larger and larger, the simulator also evolved at a rapid pace. The corresponding user interface was expanded to control the simulator efficiently. This was due to the increased facilities and commands offered by the simulators.

In 1988, it was decided that Bath University was to design and implement a real time power system simulator which would be capable of simulating the whole National Grid system. Undoubtedly, the simulator would be complex and the quantity of data produced would be overwhelming. In addition, new facilities must be provided to help engineers in studying the complex power system. It is precisely under these requirements that a Man Machine Interface needs to be implemented for that new simulator.

1.3 User Interface

Traditionally, the most important criteria in building a computer is to make it execute commands as quickly as possible. If a company could afford to own a mainframe computer in those days, it was used primarily as a number cruncher. Efforts were dedicated to make batch programs turn around quicker and hence productivity was increased. Very often, the user must adapt to the machine rather than the other way round. Needless to say, mainframe computers are not very user friendly and are very difficult to use, even to this day.

As computers get more and more power, more resources can be freed to satisfy human needs, e.g. a bit-mapped colour graphic screen can be used to display sophisticated data. At the same time, software can become increasingly complex and is therefore more difficult to use. This can be seen from the numerous commercial tutorials available to cater for the training of novice users in using various packages, e.g. word processors, spreadsheets, and desk top publishing software. Some people may regard this as a step backward. However, it should be realized that there is a totally different class of users today. While most computer operators were computer literate in the past, today's users are not. Most of them probably have no previous experience in using a computer before attempting to type out a letter with a wordprocessor. They might have little conceptual awareness in computers. Thus, as computers get more progressive, the least common denominator in the computer proficiency of users actually gets lower. The need for good user interfaces get more acute.

On the other hand, people now realize that it is important to make a computer easy to use. Hence, user interface must be integrated into the early design stage of a computer project [4]. In the past, it was more likely to be an after-thought when

the product was actually finished. Since the 1970s, much research effort has been made to investigate how a computer can best interact with the user. Window Icon Mouse Popup menu (WIMP) [1] was one of the successful concepts developed at the Xerox Corporation's Palo Alto Research Center. Since its appearance, more and more computer systems with similar interfaces have been produced. The situation was promoted by the success of Apple's Macintosh [5] in the business and education communities. Nowadays, a bit-mapped graphics screen, with a pointing device (e.g. a mouse) and a window system has been identified with a good user interface. Such a system is generally called a graphical user interface.

Although these systems might be easy to use for a novice user, they can be quite frustrating to use for an expert user. Obviously, there are different sets of constraints and requirements imposed by these categories of people. Different applications have different requirements. Different users behave differently in using the same application. Even a single user does not always act consistently. Whiteside [6] has carried out studies to show that "Interface styles are not related to performance or preference (but careful design is)". As a result, the problems of user interface design can not be solved merely by the advances in computer hardware and software. A more vigorous and fundamental study into how people behave and interact with computers to solve problems is thus more desirable, if not essential. Nevertheless, the choice of a proper interface style does in some way help the user in accepting a particular application. It is important to research into what users want in different applications and hence a suitable interface can be designed for power system engineers in using the real time power system simulator.

[1] Refer to Glossary

1.4 Graphical User Interface

In recent years, Graphical User Interfaces (GUIs) have become very popular among today's powerful computer workstations. In particular, the X Window System [7] is now recognized as the industry standard in the field of user interface design. The success of the system can be attributed to the following factors:

1. Hardware independence. An application developed on one hardware configuration can easily be ported to another. The user interface behaves consistently across a spectrum of hardware.

2. Versatile interface building tools. There is a rich library of functions and some high level software objects are available to help the programmer in building a powerful user interface.

3. Policy free. The X Window System does not impose any design policy on the interface design. Only mechanisms are provided. The programmer is free to implement whatever interface is suitable for an application.

A GUI usually presents information in the What You See Is What You Got (WYSIWYG) [2] style. Because of this, the user can understand the application better and the information produced can be easily digested. The Direct Manipulation technique helps the user further by providing better control over the application. Multiple and overlapping windows aid the user's short term memory by accomplishing an application task as quickly as possible. Hence, the X Window System as a GUI is a very powerful tool in implementing user interfaces.

[2] Refer to Glossary

1.5 About the Thesis

The design of a good MMI depends on a thorough understanding of the human behaviour and psychology. Chapter 2 addresses the issues of various human factors. A set of design "guidelines" are derived in designing a MMI. The pros and cons of what interface design system, i.e. UIMS [3] or toolkit [4] is best suitable for building the MMI are also weighed. Chapter 3 introduces the X Window System as a powerful Graphical User Interface (GUI) tool. The reasons behind why it was adopted to implement the MMI are also discussed. The computer operating system, Helios [8], is presented in Chapter 4. Together with the computer hardware (a powerful multiprocessor Transputer [9] system with a high performance VSC graphics system [10]) discussed in Chapter 5, these two chapters serve to give a technical background of the computer system that the MMI and the Simulator are operating on. The design concepts and the implementation techniques of the MMI are given in Chapter 6. As the X Window System was not designed for producing animation graphics, the method adopted to overcome the problem and the compromises made in the design are also discussed. The powerful features and some screen dumps of the MMI are presented in Chapter 7. Chapter 6 and Chapter 7 illustrate how the MMI has been successfully implemented to satisfy the requirements of a user friendly interface for a large and complex simulator. Chapter 8 discuss what further work is needed to enhance the MMI and a conclusion is made in Chapter 9.

[3] Refer to Glossary
[4] Refer to Glossary

Chapter 2

User Interface Design

2.1 Introduction

User interface design is a very difficult business. It is the combination of two distinct disciplines: psychology and computer science. A good psychologist must be sympathetic and understanding towards people while a computer scientist must be mathematical and precise. A good user interface designer must be an expert in these two different worlds. The job is made harder by the fact that the history of user interface design is even shorter than that of computer itself. Hence, little experience can be learnt from the past. Coupled with the fact that there is not a set of right or wrong guidelines in the design process, it is indeed a very difficult task to design a good user interface.

This chapter deals with the user interface design from the psychological point of view and the various human factors relevant in the design process are discussed. Any interface designer must realize what people are, what they want and how they behave, before they can be helped. Having gathered these human qualities, the designer needs to explore how a user views a computer and a program in

his own mind. Users usually construct internal models of the systems they use. Obviously, discrepancies and conflicts would emerge if the designer's model does not match with that of his user. In order to actually implement an interface, various issues must be addressed and these will be discussed later in the chapter. As far as building tools are concerned, there are generally two types available, namely UIMS [1] and Toolkit [2]. The merits of these two methods are also examined.

2.2 Human Factors

People are different. Each person has his own characteristics. It is thus impossible to categorize humans into different pigeon holes and give a label to each one of them. Fortunately, there are certain features which are unique to the Homo Sapien. It is possible to discuss human factors in general terms so that a better understanding can be gained as to how thoughts, learning and reaction to external stimuli are organized. Consequently, an interface designed for a category of users, within a certain application, can then be suitably produced. Unfortunately, humans have limited cognitive resources. It is impossible for them to explore every problem in depth as it takes too long to accomplish. Most often, humans will not bother to examine the problem. Thus, it is possible for someone to maintain inconsistent beliefs. For example, a user might exhibit non-transitive preferences behaviour. Normally, preferences should be transitive. If A is preferred to B, and B to C, then A should be preferred to C. However, it is possible that a user might prefer form-filling to command line interpreter; he might prefer direct-manipulations to form-filling. Yet, he might also prefer command-line interpretation to direct-manipulations. These inconsistent beliefs can survive within a user's mind as long as they are not weighed at the same

[1] Refer to Glossary
[2] Refer to Glossary

time. Nevertheless, gaining an insight into how humans behave does give a set of proper guidelines as what is good or bad. It just has to be borne in mind that it is not possible to satisfy all of the users some of the time; some of the users all of the time; but simply some of the users some of the time.

2.2.1 Memory Model

Humans have transducers to transmit external stimulus to the five sensory memories, namely touch, smell, sight, hearing and taste. Information is buffered in a memory for about a quarter of a second. These memories have large capacity but need refreshing from time to time if information is to be retained. This can be done by repeating the external stimulus. To remember something for a longer time, the mind must attend to a particular sense and select a sensory memory.

Once a memory has been selected, information is processed and copied to a small **Short Term Memory** (STM), also called the Working Memory, where consciousness resides and where thinking is done. The STM has only a small capacity. It is said that [11] only a total of seven plus or minus two things in the STM can be retained at any one time. If the information is unused, it is lost after 20 seconds. However, the information can be retained indefinitely by a refreshing process called rehearsal. Inside the STM, data is stored into abstractions, called **chunks**, which are arranged into a hierarchy with further sub-chunks below each chunk. Apparently, the more a chunk can abstract, the more information can be associated with it and hence more information is stored. Hence, the STM becomes more efficient.

The experience of forming a chunk in the mind is called **closure**. If people are

inhibited from forcing a closure, they will make inefficient use of STM and become liable to the risk of losing information from it as they attend to the interference. Sometimes, it is possible for a closure to interfere with other awaiting closures. If this happens, a termination error is said to have occurred. When automatic cash tellers were first introduced in the high streets, some people very often forgot to retrieve their cards after cash had been dispensed. The closure of the task of getting cash had interfered with the task of retrieving card. A better user interface was introduced to solve the problem: cash is dispensed if and only if the card has been retaken by the user.

STM is both resource and data limited. If there is not enough data given, it is said to be data limited. More resource is needed to compensate for the inadequacy of data in order to make use of it. For example, wearing a pair of glasses not only cures short-sightedness, but might actually improve hearing as some resource is called back to attend to hearing, rather than used to improve limited data, i.e. eye sight. On the other hand, if too much data is presented, the resource is stressed and data processing can fail. Thus, too little or too much data is a bad thing.

After 5 seconds residing in the STM, information is shifted to the **Long Term Memory** (LTM) which can record information indefinitely. The LTM has a very large capacity and low decay rate. Unfortunately, memory access becomes increasingly difficult without rehearsal because of interference from other memories, and it takes a relatively long time to recall information. The performance of the LTM can be improved by elaboration techniques, using mnemonics or mental imaginary. In a crowded screen with a dozen covering windows, it is common to use icons to represent temporarily unused windows. This can both save screen space and keep the user in mind what windows were there lying unused.

Both STM and LTM can suffer from the interference effect. For example, learning a new interactive system is influenced by previous experience. Such interference is called Carry-Over or Conceptual Capture. For STM, **Priming** is more acute. It is the effect of putting an idea into someone's head and make it easier for him to use. For example, although external doors should be designed to open outwards, how many times have you tried to push open an external door outside a building in order to rush inside? As a result, when a user is under pressure, he often resort to stereotyped behaviours, regardless of previous experience or training. It is thus important to create an user interface which does not violate traditional conventions and actions resulting from it must not be disastrous or irreversible.

2.2.2 Self-regulation

Humans are self-regulating creatures. They tend to adjust themselves well to compensate for external changes in order to maintain a previously balanced condition. For example, sweating is used to dissipate excessive heat so that the body temperature is maintained at a comfortable value. In designing an user interface, if a system is made easy to use, it will only be used to do more difficult things. If it is safer to use, it will only be used in more extreme circumstances. With the advances in motor cars, road accidents happen even more frequently than in the past. Hence, users are not better off in either case. It might be desirable to make an interface difficult to use in certain respect just to keep the user from becoming too comfortable. Conversely, the user should be alert in using the application in any case.

2.2.3 Flexibility

Satisficing is a form of local optimization where awareness or avoidance of the cost of establishing an overall best-choice leads to the acceptance of a good-enough choice [12]. People do not like to research into looking for an optimal solution to accomplish a task. They are satisfied with learning a basic set of commands which can enable them to get their work finished. That is why not many users like reading the user manual before using an application. Unless it is too difficult to use, they will stick to whatever methods they have accidentally discovered. That is, until disaster strike in the future. Therefore, a flexible system must permit users to adopt theoretically non-optimal strategies for particular tasks and allow different users to make their tradeoffs at different levels.

2.2.4 Skill/Rule/Knowledge

Human capabilities can be broken down into three categories: Knowledge, Rules, and Skills [13]. Initially, knowledge is gained by consciously working through a system. Later, rules are derived from experience. Improved proficiency leads to improved skill. This improvement is often achieved unconsciously throughout development. The learning curve is very steep in the beginning and gradually flattened at a later stage. Finally, a skill is acquired subconsciously but cannot be recalled voluntarily. Do you remember which foot you first raise to walk after sitting down for a while in a chair? Likewise, as a user becomes more familiar to a particular user interface, the details of how a task is accomplished are missed out. If the user has developed a bad habit at an early stage, it is perhaps too late to cure it later. Thus it is better to assist the user in the learning stage to ease the transition from knowledge to skill. Moreover, a user is much more motivated

if the transition is smooth.

2.3 Model Theories

A user uses a computer to accomplish tasks with a specific aim in mind as about which tasks should be done and some ideas as to how the tasks can be carried out. Gradually, a user develops a model of the system being used. This is modeling because humans have a natural tendency to understand, to model their environment: both to be able to control it and to be able to think about it. Such models can be categorized as :

1. **specific and analytic**

   the computer works in this way or that way.

2. **metaphorical**

   the computer works like a typewriter.

3. **teleological/anthropomorphic**

   the computer does not like this, it wants to do that.

The nature of these models as devised by the user depends very much on the training and education background acquired in the past as well as previous experience with a similar system. To some extend, the models are also governed by the designer of the computer and the user interface.

On the other hand, computers also have models of their users, derived either implicitly or explicitly by their designers. A designer usually defines a **Canonical Model** to describe the potential users of his system. The term canonical

means "according to formula". For example, the designer of a spread sheet may assume the user would never have a database larger than 1 GBytes. The model is implemented in the form of a program which a user interacts with.

Intrinsically, models/programs are not equivalent. Indeed, it would be incredible if the designer's thought matches exactly with that of the user. In order to ease the problem, it is conventional practice to improve the information bandwidth of the user interface so that each (the computer and the user) can gain a better picture of the other. Practically, this is done by training courses, better documented manuals, or a better user interface.

There are two ways in which a user interface can be improved:

1. Increasing the coherence between the models and the programs. However, it is neither possible nor desirable to have two identical models. The canonical model must have some advantages over the user, i.e. speed, resolution and persistence.
2. Decreasing reliance on the coherence. This can be done by increasing the information capacity of the user interface. Unfortunately, human brains have a limited capacity for storing and processing information.

It appears that a compromise must be reached between the two methods

2.3.1 Bottlenecks

Other than the mismatches between models and programs, another obstacle which affects the communication between a computer system and its user is the

bottleneck. Bottlenecks exhibit themselves in three ways:

1. **The Von Neumann Bottleneck**

   A computer has limited capacity for processing and transferring information. If there is too much data waiting to be dealt with, the response time taken for a computer to service a data request will be lengthened.

2. **The Interface Bottleneck**

   - The computer input/output Bottleneck.

     For example, the screen is not big enough or of high enough resolution.

   - The user input/output Bottleneck.

     The user's senses are not fast enough to response to the computer. For example, playing a video game.

3. **The Mental Bottleneck**

   The information waiting for the user is just too much for him to digest. The user can not remember all the details displayed on the screen. He might even find it difficult to grasp the significance of a piece of data.

One way of widening bottlenecks is to use **Filtering**. A computer can use cache memory to store only that most frequently accessed data so as to avoid the slower, though larger external memory. Restricting the amount of information given by the user gives the computer a chance to deal with the input. Temporarily, irrelevant information is not displayed on the screen, assuming that it can be accessed at a later stage. Advice are available to the user to assist him in understanding the output.

2.3.2 Magical thinking and Non-Determinism

When a model is inadequate, a user develops magical thinking, so as to compensate for the missing data. This could happen if the user does not have an adequate knowledge of the system. Alternatively, if a system is intrinsically unpredictable, it appear to be non-deterministic, or behaves randomly, to the user. For example, in a windowing system, some pop-up menus are created when the user is idle so that the system resource is not wasted. Thus, when a menu is popped up, it could do so either very quickly (created already) or slowly (not yet created). The result is totally undetermined. If the user is not aware of this resource exploitation policy, he will start to create a mythical model on his mind to explain the system's inconsistent behaviour. If, in the beginning, he found that a menu is popped up quickly when the left mouse button is pressed (by pure coincidence), he might think that the left button is a sort of accelerator for popping up menus. Just imagine what he might think when pressing the same button has no effect, or even slows down the popping up of a menu.

Hence, a technical view of the system gives the user conceptual power. However, it is quite unreasonable to assume that the user has to learn how a particular application is implemented before using it. Why should the user know that the programmer has allowed for only a maximum of 100 lines for buffering so that the system would crash when 101 lines of data are entered? One way to minimize the damage is to provide safety checkings in the codes. If a user has entered 100 lines, issue a warning and stop accepting any more lines. Another way is to make the system behave consistently, even at the expense of efficiency. For example, all pop up menus can be created once at the beginning of an application.

2.3.3 Projection and Transferrance

Having developed a system model, the user will tend to interact with the system as if it were part of the model. The user will project wishes on to the system and expects the system to behave in the same manner.

Sometimes, a user does not only make a **Projection** on his own model, but one from someone else. **Transference** is said to have taken place. In user interface design, this means that a user transfers models learnt with other systems on to the present one, thereby making the interface design more difficult. If a system is totally new to a user, this can be very frustrating. However, if the system looks similar to a previous one, the user might form a distorted model of the new system. The user can either make a lot of mistakes or be unable to understand the new system at all.

2.4 Interaction Styles

The way in which a user interacts with a computer is dictated by the interaction style that the user interface has imposed upon the application. Traditionally, computer programs use **Modes** to simplify applications. However, the benefits of using modes are superseded by their short-falls. Instead the WYSIWYG [12] style of presentation has gained popularity over the years. This leads to the introduction of **"Direct Manipulation"** [14], which has now become the norm in modern user interfaces. The reasons behind its success and its effects on user interface design are now examined.

2.4.1 Modes

A mode is defined as "The information in the computer system affecting the meaning of what the user sees and does" [12]. It is used to ease the communication bottlenecks between the user and the computer as non-essential information is concealed from the user. Effectively, the communication bandwidth is widened by binding more meanings to a particular key on the keyboard when it is in different modes. More commands can be accommodated as a result. The program can also be made simpler from the designer's point of view. Another advantage of using modes is that the user is protected from making fatal mistakes. For example, a file can only be destroyed if the user has pressed a certain key and the system has toggled into another mode. It prevents the user from deleting the file accidentally.

However, modes can be a great disadvantage to the user as information is hidden. This could be either forgotten or not noticed by the user. The system gives the user no clue about how it got to its present state or does not tell the user enough for him to be quite certain what the current state is. Obviously, this can cause a lot of confusion for the users.

George Polya [15] states in his **Principle of Non-Sufficient Reason** that "if you do not have sufficient reason to doubt things are different (when solving problems) treat them as the same". This effectively means that a system should not work differently unless it gives the user sufficient information to tell the modes apart. Otherwise, the user can not be certain of the present system state.

Hick's law [16] states that: "The user's decision time is proportional to the logarithm of the number of choices known to be open to him". So, it is too easy to forget what state of a system is in and how it will interpret commands: the

more modes there are, the more likely the system will use an inappropriate mode. Even if the user knows exactly which mode a system is in, his reaction time will increase.

Karl Popper's **Berkeley's Razor** [17] states that, "All entities are ruled out except those that are perceived". If "entities" equates with "information", that rules out hidden information that the user can not perceive. So fewer modes makes a user see clearer in a system.

"Out of sight, out of mind" Modes should be avoided as much as possible.

2.4.2 What You See is What You Got

One way to use fewer modes is to use the WYSIWYG [12] style of presentation. In WYSIWYG, there is a simple relation between information transmitted from the computer to the user and the results obtained, i.e. what the user can see (information transmitted from computer to user) equals what the user gets (information transmitted from computer to results). If the system has no hidden information, the user can manipulate what is seen, without fear of unseen side-effects. The system model can be understood and relevant parts can be readily memorized by the user without prior system experience. The user may generalize his knowledge and is confident as to what has happened. The designer can use this principle to meet clearly definite user expectations with specific techniques.

2.4.3 Dialogue Models

A dialogue model is an abstract model that is used to describe the standard of the dialogue between a user and an interactive system. There are three types [18] of dialogue models:

- **Transition Network**

  The progress of a dialogue can be viewed as a series of transitions from one state to another. The dialogue may be in a particular state awaiting input from the user, and it will progress to one of the several next states depending on the nature of the input received. This can be represented as a transition network. Each state is represented by a node. Transition between nodes is indicated by directed arcs connecting the two nodes; a label on the arc indicates the condition under which it is traversed. The Question and Answer structure dialogue is one example.

- **Grammars**

  The dialogue between a user and a computer system can take the form of context-free grammar. It is similar to the natural language, spoken between two people. In the case of human-computer interaction, two distinct languages are employed: the user uses a language to enter commands to programs, and the computer uses another to communicate the results of these commands.

- **Events**

  This model is based on the concept of an input event that is found in a number of graphics packages. In these packages, the input devices (e.g. mouse) are treated as sources of events. Each input device generates one or more events when the user interacts with it. An event has a name or number

that indicates the nature of the interaction, plus several data values that characterize the interaction. For example, a mouse movement generates a series of events recording the x, y coordinates of the movement. Depending on whether any button is pressed or released, further events are generated. Typically, a pop-up menu will then be displayed, prompting for further user actions or displaying some system messages resulting from the events.

The merits of a dialogue model can be established by evaluating it according to the following qualities [19] :

1. **Naturalness**

   The model does not cause the user significantly to alter his normal approach to the task in order to interact with the system.

2. **Consistency**

   Expectations that the user built up using one part of a system are not violated by the idiosyncratic changes in the convention used in another part.

3. **Non-redundancy**

   The user does not have to input more than the minimum amount of information for the system's operations. This has the merit of making the interaction both quicker and less prone to errors.

4. **Supportiveness**

   The model should provide assistance to the user in running the system. The quantity and quality of instructions, the nature of error messages and the use of positive feedback for the user's instruction are measures of a good dialogue model.

5. **Flexibility**

   The model should cater for or tolerate different levels of user familiarity and performance. Such flexibility implies that the dialogue is able to adapt either its states or the input it will accept. For example, short-cuts should be provided for an experienced user so that he can avoid issuing simple, but tedious, commands.

6. **Less burden on the user's STM**

   As humans have limited STM, the structure of a dialogue should be organized in such a way that related commands are grouped together so that closure can be achieved more frequently. In this way, the memory demand on a user can be reduced to a minimum.

All the three dialogue models, Transition Network, Grammars and Events, have pros and cons in each of the criteria. However, it is argued [18] that it is possible to create a user interface that can only be described in the event model, but not in the other two models, i.e an event model has more descriptive power. Hence, it should be formed as the basis for the internal representation used in a system. Although it might be necessary to use the other two models occasionally, it is only necessary to implement run-time support for the event; the other two can then be translated into this form.

2.4.4 Direct Manipulation

Having established a WYSIWYG presentation style and event dialogue model, it is only natural to combine the two and implement the **Direct Manipulation** interaction style for a user interface. The objects of interest are directly visible and direct reversible incremental actions can be applied to them whenever the

user feels appropriate. Complex command language syntax and "hard to digest" data are replaced by direct manipulation and pictorial objects respectively. It is usually implemented as a graphic system with pictures or icons to represent objects and data. The user uses a mouse pointer to manipulate the objects directly. Commands are usually organized and displayed inside a pop-up menu and the user can use the mouse pointer to select an option.

Another advantage of direct manipulation is that the feedback is immediately visible. A user can then determine if his last action has the desired effect. If not, he can easily reverse the process and issue another command. The visual feedback enhances the user's learning process enormously. A user becomes more confident and feels that he is in control of the system. A beginner can learn the basic functionalities of the system very rapidly.

Shneiderman [19] suggested an interesting example to illustrate the power of direct manipulation. In it, a driver uses the steering wheel to turn a car in whichever direction he wants. He can see the outside world clearly and can adjust to it as soon as it is necessary. Then, imagine what it would be like if he can only type a command line into a keyboard in order to control the car and the result is printed out as a sequence of numerical data!

Direct manipulation has been a well researched subject [20–28]. Results have led to successful learning methods and better user interfaces. However, direct manipulation does have some drawbacks as well [19] :

1. Spatial or visual representation is not necessarily an improvement. The content of graphic representation is a critical determinant of utility. The wrong information, or a too clustered presentation can lead to greater con-

fusion [19].

2. The user must learn the meaning of components of the graphic representation. It takes time to learn. Besides, graphics do not have universal meanings. For example, a Danish or Thai trash can will look different from an American one [29,30].

3. Graphic representation may be misleading. The user may rapidly grasp the analogical representation but then make an incorrect conclusion about permissible action. For example, the waste bin in most computer systems behaves more like a paper shredder as the action of dumping a file there is irreversible.

4. Graphic representation may take excessive screen display space. Thus the computer input/output bottleneck is further squeezed.

5. For an experienced user, it becomes more tedious to move around the system using a mouse pointer than typing direct and complex, but powerful, commands.

There are some methods which can reduce some of these problems:

The first provides a text label in the user's own dialect besides a graphic object. Though a trash can looks different in each country, the meaning of the pictogram can be understood by the user. Alternatively, the graphics can be better documented.

The second provides short cuts for the experts so that it is not necessary to go through the whole sequence of commands that a novice user has to invoke in order to accomplish a complex task.

2.5 Colour

In the context of user interface design, the effective use of colour can convey much information to the user and help him learn to use the system. Conversely, a bad choice of colour scheme can seriously confuse the user and make the system totally unusable. Hence, a basic understanding of the colour theory is vital to a successful user interface design.

Information presented in colour is easier to identify and is easier to distinguish in the periphery of the visual field. It is also learnt faster and remembered longer than information coded by shape, size and brightness [31]. It is possible to make use of colour and apply it to:

1. visually group objects together in order to bring a sense of order to a complex scene. Repeated occurrences of an object in a single scene or across multiple scenes can be collected together.
2. differentiate an object from its surroundings and make it stand out from the background.

Effectiveness of the choice of colours depends on the types of coding chosen for the application and on the expectation of the user. There are two methods of coding for colour:

1. **Nominal Codes**

   associate a set of unrelated colours with certain states of a system, e.g. the colour scheme used to distinguish a live line (red), a neutral line (brown) and an earth line (yellow and green) inside a domestic three pins plug.

2. **Ordinal Codes**

   imply an ordering of the values of one or more variables by arranging colour in a specific order, e.g. the voltages of busbars within an electricity transmission system can be colour coded according to their magnitudes.

Although it is easy to design a colour system with a computer, it is very difficult to make effective use of it. The designer lacks the knowledge as how colour can be applied in a system. MacDonald [32] compares that situation with the case of Do It Yourself type of work. It might to easy to collect all the essential parts of a wardrobe, assembling them together in a pleasing and practical structure is another matter. Over the years, a lot of research has been spent on studying colour. Some guidelines can be drawn from some of the previous work [32–34]:

1. Association of Colour

   Users can be conditioned into having certain expectations about the meaning of colours, due to their association with natural, emotional, cultural or technical contexts.

   - colour ordering may correspond to natural phenomenon, e.g. the radiation of a black body goes from black-red-yellow-white.
   - colour has the ability to evoke an emotional response or to trigger memories, e.g. warm hues (red, orange, yellow) can imply action or comedy, cool hues (green, blue, grey) can imply passivity or sadness
   - colours can have a cultured or geographical association. In Western culture, red = stop = danger, while green = go = safety.
   - colours can have technical or contextual associations. In the display of medical body-scan images, it is customary to indicate healthy tissues by red or yellow and diseased areas by green or purple.

2. Information Coding Principle

   Coding can be used to increase the information content of the display or improve the display by making it easier to interpret.

   - colour can be used in conjunction with other visual attributes such as shape, size, brightness, etc, to help the user in interpreting the display more positively by reinforcing the information, and also help those users with impaired colour vision.
   - the number of colours used on a display should be restricted to about 12, or as few as 5 in some critical cases, if they are to be identified unambiguously under a wide variety of viewing conditions.
   - colour codings should be used to group related items together, even though they might be physically separated. Conversely, unrelated adjacent items can be differentiated by applying a set of distinct colours.
   - colour changes can be used to dynamically identify states changes.

3. Performance of the Human Visual System

   The biological structure of the human eyes dictate how different colours are reacted to under different environments.

   - the eye is most sensitive to yellow-green and least sensitive to red and blue, under normal daylight conditions.
   - the luminance ratio between a character and the background should be about 10-1, for optimum legibility. Dark characters on a light background need to use bolder fonts than light characters on a dark background.
   - displays of large areas of saturated colour may lead to visual fatigue or confusion after prolonged viewing. Also, after-images of strong colours may appear in the complementary colours.

- a combination of intense primary colours red/green/yellow/blue, should be avoided as they can cause unpleasant "variations" in the image. Colour combinations that contrast in lightness as well as hue should be used in order to assist the perception of edges.

2.6 User Interface Development Tools

Creating a good user interface for a system is very difficult. There are no guidelines or techniques which guarantee that the software will be easy to use and user interface designers have generally proven to be poor at providing interfaces that users like. It is important to realize that an application's interface can account for a significant fraction of the code. Surveys of artificial-intelligence applications, for example, report that 40 to 50 % of the code and runtime memory are devoted to interface aspects [35]. The production and maintenance costs are high for systems using interactive graphic interfaces, demanding up to 50 % of the entire system development time and as much as 60 % of maintenance costs [36]. Therefore, tools and methods are critical to the development of quality interfaces [37]. There is great interest in developing tools to help design and implement interfaces.

2.6.1 Separation of Interface and Application

There are two reasons for separating user interface from application code:

- to enhance reuse
- to provide high quality interfaces

In the past, interfaces to applications have been developed using ad hoc and low level techniques. The interface is frequently buried within the application code in a manner which both hinders maintenance, and makes the reuse of the methods within a different application time consuming [38]. Inconsistent interaction methods and badly structured systems are the direct results. Separation lets specialists develop the user interface and the application independently. It also promotes interface consistency across applications, and let designers add or combine application functions in new ways. It is observed that consistency is a critical area in user satisfaction [39].

The earliest approach to separation was to maintain a strict division of responsibility between the user interface and the application: the application did the work and the user interface communicated with the user. However, the user interface must have sufficient access to application internals to keep the user aware of application semantics (the application objects and operation) [40]. For example, a good user interface protects the user from invoking an operation that cannot be executed successfully. On the other hand, if the user interface has too much knowledge of the application, the distinction between user interface and application blurs.

Clearly, a good user interface has some knowledge of application semantics. The questions are how much knowledge is necessary and how should it be captured?

To address theses problems, it is necessary to experiment with different interfaces for different application as quickly as possible, i.e. **rapid prototyping** is needed for evaluating the performance of a particular interface. There are generally two types of development tools available for the user interface designers to create user interfaces rapidly, namely User Interface Management Systems **UIMSs**, and

Toolkits, which are the subjects of the next section.

2.6.2 UIMS and Toolkit

The advantages of using user interface tools are:

1. Better Interface
    - rapid prototyping and implementation
    - easier to incorporate changes during design stages
    - one application can have many interfaces
    - more effort can be expended on the user interface tools than may be practical on any single interface because the tools are reusable
    - different applications can have a more consistent interface because they have been created with the same user interface tools
    - easier to investigate different styles for an interface, thereby providing a unique look and feel for a program
    - easier for many specialists to be involved
2. User interface is easier to design and economical to maintain:
    - code is better structured and more modular
    - higher reliability because code is created automatically from a higher level specification
    - interface specification can be represented, validated and evaluated more easily
    - device dependencies are isolated in the user interface and the application is portable across a range of computer hardware and software

Although user interface development tools are generally good at defining the static layout of an interface through direct manipulation, they are not very useful in helping to create the user interface's dynamic behaviors. [41].

UIMS

Lee [41] defines a UIMS as "a software architecture in which the implementation of an application's user interfaces is clearly separated from that of the application's underlying functionality. The separation is true both physically (separate code modules) and logically". In practice, a UIMS is an integrated set of tools that help a programmer to create and manage many aspects of interfaces.

The UIMS helps both with designing and implementing the interface and so encompass a broad class of programs [41–48]. As many as six main different categories of UIMS, classified according to implementation methods, have been reported [42].

Toolkit

Lee [41] defines a toolkit as "a library of components like menu, command buttons and scroll bars. Programmers access the interface system through their operating system's standard language/library calling process." There are three kinds of toolkits [42] : The first uses a collection of procedures that can be called by the application program, e.g. Macintosh Toolbox [5]. The second uses an object-oriented programming styles with inheritance, which makes it easier for the designer to customize the interface techniques, e.g. MIT's X Window Sys-

tem Manager [49]. The third adds design constraints to object-oriented toolkits. Constraints let the designer specify relationship among objects and have the relationship maintained by the system, e.g. Grow [50], Coral [51].

Toolkit-UIMS Comparison

Over the years, the toolkit has been more successful than the UIMS [41,52]. The growing success of MIT's **X Window System** [7] is a very good example. The reasons behind this can be explained by the fact that the UIMS is inferior to the Toolkit in the following respects:

- UIMS has a limited range of interfaces. Since it allows interface specification at a higher level, the range of interfaces that can be created is limited.
- Some UIMS rely on obscure techniques, e.g. a special interpreted specification language. This is unfamiliar to programmers and interface designers. Moreover, the language is usually inferior to established general purpose language and hence the performance is degraded.
- UIMS is inadequate in direct manipulation interfaces. The strict separation of application and interface codes usually results in a low-bandwidth communication between the two.

2.7 Conclusions

This chapter has described most of the design aspects of a good user interface. The theories, guidelines, models and methods developed here will form the tools

for building up the Man Machine Interface and the reader has been given the philosophy behind the design criteria of the research project.

In summary, a good interface should be designed in the following manners:

1. The limitation of the Short Term Memory and Long Term Memory of humans should be understood.
    - The closure of a task should be completed rapidly to avoid interference.
    - The amount of data presented to the user must be neither too much nor too little. Filtering can be used to reduce the information flow from the computer to the user.
2. The interface should not be made too easy to use so that it is not pushed to its limits.
3. Non-optimal strategies should be allowed to let the user to accomplish particular tasks. The skill levels and preferences of different users should be catered for in a single interface.
4. A smooth learning transition, knowledge-rule-skill, should be provided to motivate the user .
5. The conceptual model of the interface should be simple so that the user can understand the system and make fewer mistakes.
6. The interface should behave consistently so that the user's Magical Thinking about the system is eliminated.
7. The interface should not violate conventional wisdom because the user might bring forward experience learned in another system to the new one.

8. Mode should be avoided. Instead, event driven direct manipulation and WYSIWYG should be used to provide:

    - Naturalness
    - Consistency
    - Non-redundancy
    - Supportiveness
    - Flexibility
    - Less burden on the user's Short Term Memory

9. Colour should be used effectively to:

    - associate and differentiate relationships between objects.
    - display pleasant looking graphics to enable the user to understand the data displayed.

10. Separation of Interface and Application should be maintained to enhance portability of the Interface.

11. The toolkit should be used as the main tool as building the interface as it is more flexible and powerful.

Chapter 3

The X Window System

3.1 Introduction

The advent of powerful microprocessors, bit-mapped graphics and colour display has allowed graphical user interfaces (GUIs) to become the norm in the field of man machine interface design. In the past few years, a number of powerful GUIs have been introduced which take advantage of the advances in computer hardware. For example, Macintosh interface [5], X Window System [7], Windows [53], Open Windows [54], Presentation Manager [55], DEC Windows [56] and GEM [57] are very popular among today's most powerful computer workstations. Basically, a GUI system can be identified as one which has the following characteristics:

1. a pointing device, e.g. a mouse.
2. on-screen menus that can appear or disappear under user control.
3. windows that can graphically display what the computer is doing.
4. icons that can represent some application objects, e.g. files, directories.

5. dialogue boxes, buttons, slides, and a variety of other graphical widgets that let the user tell the computer what to do and how to do it.

Not every GUI has all of the above features. For example, some GUIs do not use icons. Some GUIs require mice while others might use trackballs or keyboards as input devices. However, all GUIs allow the user to interact with an application by direct manipulation [1] The apparent success of these GUIs is due to the fact that a GUI is easy to learn and use for the novice user while at the same time complex information can be presented to the more experienced user.

Despite the competition from various GUIs, the X Window System, or simply "X", has become more and more popular among the powerful computer workstations. Some of the success can be attributed to the powerful features and design philosophy of X. This chapter introduces the history, design philosophy, software hierarchy and general features of X.

3.2 History

X was developed at MIT in the early 1980's. It was the result of two independent groups at MIT having simultaneous need for a window system. The Angus System [58] requires a debugging environment for multiprocess distributed applications. A window system was regarded as the best solution. Project Athena [59] needed to integrate a large quantity of workstations, from a variety of hardware manufacturers, with bitmap displays.

[1] The importance and advantage of using direct manipulation as an interface style are discussed in Chapter 2: User Interface Design.

The W Window System [60] was developed at Stanford University. It was in turn produced as an alternative VGTS [61] for the V System [62]. The W Window System produces graphics windows based on a simple display-list mechanism, with limited functionality. A Unix version of the W Window System [63] was acquired by MIT and X was later produced to overcome some of the shortcomings of the W Window System and satisfy the local requirements at MIT.

In 1986, version 10 release 4 of the X Window System was released by the Athena team. Later, in January 1988, a consortium with most of the leading workstation manufacturers was formed by MIT to develop X further and had it adopted as an ANSI standard. Since then, X has been implemented on a variety of hardware and operating systems. The X Window System running on the Helios operating system [8] was based on version 11 release 2 of the MIT X Window System. The MMI was implemented using the Helios X Window System [64].

3.3 Design Philosophy

The design philosophy of the X Window System can be revealed by examining the set of requirements laid down by the early developers [65] :

1. The system should be implementable on a variety of hardware with different bitmap displays and input devices.

2. Applications must be device independent so that different applications behave consistently across different hardware. If hardware is upgraded in the future, an application should be immune to the changes.

3. The system must be network transparent so that an application is allowed to make use of some remote resources across the network which is made up of a variety of machines or computer operating systems. This is especially useful when running a complex application. The calculation part of the application can be run on a remote and powerful mainframe computer while the graphics display is shown locally on a smaller but cheaper microcomputer.

4. Multiple applications must be allowed to be displayed concurrently so that the user can make use of different applications at the same time.

5. Many different applications and management interfaces must be supported. The system must be policy free and should not impose any restriction upon an application. Instead, a rich set of interface components and control mechanisms should be provided for the interface programmers. The "look & feel" and interaction style is up for the individual programmer.

6. Overlapping windows, including output to partially obscured windows, must be supported so that a great variety of interface can be implemented.

7. A hierarchy of resizeable windows and many windows that can be brought up simultaneously, should be available. These features are useful to an application with a lot of graphical output and the programmer is freed from the burden of implementing window clipping and input control.

8. High-performance, high-quality support for text, two dimensional graphics and imaging should be supported. Programmers can build up different sets of graphics models on top of the X Window System in order to cater for different application requirements.

9. The system should be extensible so that it is relatively easy to add extensions to it which are suitable for a particular application.

3.4 Software Hierarchy

The X Window System is based on the client/server model. This is in answer to requirements two and three discussed in the last section. Fig. 3.1 shows the system structure of a typical X Window System.

3.4.1 X Server, X Client and X Protocol

X achieves device independence by providing a complete virtual "display" in software. This includes the keyboard, mouse (and/or other pointing devices), screen or screens and the controlling processor. An X server is a software program used to control the display. It contains the main device-dependent part of the X Window System and performs the following tasks:

1. allows access to the display by multiple clients.
2. interprets network messages from clients.
3. passes user inputs to the client by sending network messages.
4. draws two dimensional graphics.
5. maintains complex data structures.

An X client (application) is a program running on the X Window System making requests of the server to produce graphics and receive user inputs.

Multiple clients can have connections open to a server simultaneously and a client can have connections open to multiple servers at the same time. The

client and server might be resident on the same computer, or they may be very widely separated but connected to a network. The essential tasks of a server are to multiplex requests from clients to the display, and demultiplex keyboard and mouse inputs back to the clients. A client communicates with the server by sending packets of instructions, obeying a set of network rules, the X Protocol [66]. The communication takes place asynchronously and is dependent on the user's inputs, i.e. it is event driven.

Typically, the server is implemented as a single sequential process, using round-robin scheduling among the clients [65]. The block- stream X Protocol is layered on top of a reliable duplex (8-bit) byte stream to facilitate client/server communication. The protocol requests generated by a client are variable-length data packets, followed by a 16-bit field specifying length, and one or more bytes of additional data. The added data might be numeric parameters and coordinates, text string to be printed, or raw bit-map data in scan line order.

3.4.2 Xlib, X Toolkit, Intrinsics and Widgets

A client program is usually implemented using a library of more than two hundred function calls and macros, the Xlib [67], which provides a procedure interface to the X Protocol. A client calls procedures in the Xlib to send window management and drawing requests to the server. The server sends event notifications to the client in response to user actions and screen geometry changes. The Xlib queues events and packages them into a record structure. An application periodically polls the library for the next event.

The Xlib provides a powerful low-level interface but this flexibility introduces a

major drawback: it is hard to write even a simple program. Therefore, the X Toolkit (Xtk) [68], was introduced to:

1. reduce the programming effort needed to write an X application. For example, a simple program which requires forty executable statements to implement on the Xlib, took only just five using the Xtk [69].

2. allow user customization. User preferences, e.g. colours, fonts, ... etc., can be specified in just a few lines of text in a user preference file.

3. cache data on the application side and minimize the latency introduced by the round trip queues as an application makes a request to the server.

As X is policy free, it recognizes that no single comprehensive set of user interface tools is likely to be acceptable for standardization. In order to maximize the utility and acceptability of the new interface library, Xtk has been divided into two separable pieces:

1. The Intrinsics Layer. This is a mostly policy free foundation upon which widgets and applications are built. The layer contains functions and data structures to:

   - create, organize and destroy widgets.
   - negotiate over screen real estates when a widget changes size.
   - implement more sophisticated widgets built upon a few basic ones.

2. The Widget Set. A widget is a user interface component implemented using calls to the Intrinsics and Xlib, e.g. a scroll bar. The Athena Widget Set [70] was distributed to serve as an example of how commonly seen user interface

components can be written. Many applications use only basic widgets; a few supplement these with application specific widgets. It is possible to design different sets of widgets based on the same Xtk. Hence, the "look and feel" of an application can be determined by the choice of a widget set.

3.5 Features

3.5.1 Events and Buffering

An event is a packet of information generated by the server when certain actions occur. Events can be generated in the following ways:

1. by the input devices. For example, moving the mouse pointer can generate tens of events in a row.
2. by the changes in window status. For example, when part of a window is obscured by another one, an event is generated.
3. by the clients connecting to the server. For example, a client might generate an event to notify another client in order to initiate the transfer of data.

The server maintains one event queue, on which all events are placed. Different types of events can be selected for each client. The Xlib maintains one event queue for each client, on which the selected events are placed. When an event is generated, it is placed on the server queue. Periodically, the events in the server queue are transferred over the network to the Xlib queues. Thus, it is possible that more than one client will receive a copy of the same event data if they each

select it. As the Xlib event queues are buffered, it is possible that an user action will not get noticed by a client if there are too many events queued. Likewise, a client's requests are queued in its Xlib output queue and sent contiguously to the server once in a while. Thus, delays often occur. For instance, the drawing requests produced by a client will not appear on a window until the Xlib output buffer queue is flushed and the requests processed by the server. Functions are provided to let a client to manipulate its input and output event queues so that it can:

1. search the input event queue sequentially for a particular type of event.
2. remove a particular event from the input event queue which might be pushed back for later use.
3. wait for the next available event from the input event queue.
4. clear the input event queue by throwing away all the events.
5. flush the output event queue and force all the requests to the server immediately.

As events can arrive in any order, the structure of code used to handle them is thus predetermined. Every program contains an event loop in which each event is received and processed. Depending on the types of events received, the program takes the corresponding actions. Each event is accompanied by a block of data which can be examined to obtain the necessary information associated with the event.

3.5.2 Windows

An X server controls a bitmapped screen. In order to make it easier to view and control many different tasks simultaneously, the screen is divided up into smaller areas called windows. A window is a rectangular area that can display graphical outputs and receive user inputs. Windows on the screen can be arranged so that they are visible or so they cover each other completely or partially. There are a few characteristics of windows:

1. The hierarchy of the windows is arranged like a tree. Hence, each window has a parent, except the very first window, the root window, which is created by the X server at start up. Though a child window may be positioned partially or completely outside its parent window, all graphical outputs that are outside the parent window's boundary are clipped.

2. A window has a position, which locates the upper left corner relative to its parent's corner, a certain width and height, and usually a border. Since several windows may have the same parent, a window must also have a stacking order among its siblings to determine which window will be visible if they overlap.

3. A window has characteristics referred to as depths and visual types, which together dictates its colour characteristics. The depth is the number of bits available for each pixel to represent colour (or grey scale). The visual type represents the way pixel values are translated to produce colour or monochrome output on the monitor.

4. A window can either be InputOutput or InputOnly. InputOutput windows may receive inputs and may be used to display outputs. InputOnly windows can only receive inputs.

5. A window has a set of attributes which control many aspects of the appearance and response of it. For example, the colour or pattern used for the border and background of a window are determined by the attributes.

6. The graphics content of a window are not stored by the X server. A client has to be able to redraw the content of an obscured window when it becomes visible again. Otherwise, the content will be blanked.

3.5.3 Window Manager

The X Window System provides only the mechanism for implementing an user interface, not the policy for determining how the interface should behave. Under X, the policy can be provided by a separate program called "Window Manager", which is just an ordinary client program. It should be noted that it is not absolutely necessary to have a Window Manager present to supervise clients. In fact, the clients should behave consistently regardless of whether a Window Manager is present in the system.

Client programs have to negotiate with the Window Manger, which is responsible for determining all matters of how the screen should be used. The clients should offer the Manager "window hints" of their wishes. The Manager can then use any available algorithms to satisfy these requests as fairly as it can.

A Window Manager can be used to emulate other window systems. For instance, it is possible to make an X client to look like an application running under the Presentation Manager [55]. Thus, the "look and feel" of an application can be determined by the choice of a Window Manager.

3.5.4 Resources

In order to reduce network traffic, it is necessary to restrict the flow of data between the server and the clients. The X Widow System uses an integer ID number to identify a resource which can be a window, cursor or font, ... etc.

Whenever an operation is to be performed on a window (or any other resource), the ID of the window is used in one argument to the routines. Hence, only a single integer representing a data structure is sent over the network with an Xlib routine call, instead of the entire structure.

3.5.5 Colour

For each coordinates on a two dimensional screen, an N-bit pixel is stored. The number of bits in a pixel value and how a value translates into a colour is hardware dependent. One bit per pixel is used for the monochrome and between four and twelve bits per pixel are used for the pseudocolour hardware. Each pixel value is used as an index into a colour map and obtains red, green and blue (RGB) intensities. The colour map can be changed dynamically, so that a given pixel value can represent different colours over time.

X provides an interface for the programmer to write applications covering the spectrum in a consistent manner. In addition, multiple applications can coexist within a single colour map so that they always show true colour on the screen. Hence, pixel values are not coded explicitly into applications and the server is responsible for managing the colour map and colour map allocation is expressed in hardware independent terms.

There are two ways to obtain pixel values by a client. In the simplest request, the client specifies RGB values, and the server is responsible for allocating an arbitrary pixel value and sets the colour map so that the pixel value represent the closest colour the hardware can provide. A pixel value can be shared by multiple clients and the colour map entry for it cannot be changed by the client. The server also provides a colour database that clients can use to translate string names of colours into RGB values tailored for the particular display. In this way, an application is protected from variation in colour representation among displays.

For the second request, writable map entries can be allocated by the server for the client. There are three common uses of this request. One is simply to allocate a number of "unrelated" pixel values. A second use is in imaging application, where it is convenient to be able to perform simple arithmetic on pixel values. A third form of allocation arises in applications that need some form of overlay graphics so that it is possible to draw and then erase graphics without disturbing existing window contents.

3.5.6 Graphics and Text

Graphics operations in X are expressed in terms of relatively high level concepts, e.g. line, rectangles and fonts. This promotes device independence as well as reducing network traffic.

There are two forms of offscreen images supported in X: bitmaps and pixmaps. A bitmap is a single plane (bit) rectangle. A pixmap is an N-plane (pixel) rectangle, where N is the number of bits per pixel used by the particular display. Arbitrary

size of bitmaps and pixmaps are allowed as long as the computer memory is large enough. Bitmaps are primarily used as masks in clipping graphics. They are also used to construct cursors and icons. Pixmaps are mostly used for storing frequently drawn graphics and as temporary backing store for pop-up menus.

All graphics and text requests include a logic function and a plane-select mask to modify the operations. Given a source and a destination pixel, the function is computed bitwise on the corresponding bits of the pixels, but only on bits specified in the plane-select mask. In the simplest case, a single source pixel is taken to combine with every pixel in a rectangular region of a window. This is used to fill a region with a colour. Depending on the logic function or masks, different effects can be achieved.

X provides functions to draw arbitrary combination of straight lines and curved segment. A line has attributes to determine how it should be drawn, e.g. solid or dashed. For high performance text, X provides direct support for bitmap fonts. An application can use an arbitrary number of fonts, provided that there is enough computer memory. To achieve this, the client specifies a font name and make a request to the server which then caches the font information on the server side.

3.5.7 Input Devices

The X Protocol is designed for use with a mouse having up to three buttons. An application can selectively receive events on the movement of the mouse and whether each button is pressed or released. Each event contains the current mouse coordinates, the current state of all button and modifier keys, and a timestamp

when the event occurs.

Clients can define arbitrary shapes for use as mouse cursors. A cursor is defined by a source bitmap, a pair of pixel values with which to display the bitmap, a mask bitmap that defines the precise shape of the image, and a coordinate within the source bitmap that defines the "hot spot" of the cursor. A window is said to contain the mouse if the hot spot of the cursor is within a visible portion of the window or one of its subwindows.

For the keyboard, a client can selectively receive events following the pressing or the release of a key. In order to promote portability, a key press is translated to a key event first. Each key event contains the keycode of the key that was pressed. In order to bind more meanings to a key press, a mask is used to indicate which modifier keys and pointer button were being held down just before the event. A modifier key is a key like Shift or Control that can modify the meaning of a key event. The keycode for each physical key never changes on a particular server, but the key with the same symbol on it on different brands of equipment may generate different keycodes. Clients then translate the key event into a keysym which is a defined construct that corresponds to the meaning of a key event. Finally, a keysym can be translated to its ASCII value.

3.6 Conclusion

This chapter has described the history, design philosophy, software hierarchy and general features of the X Window System. The reasons why the MMI is implemented using X can be categorized as follow:

1. X is highly portable across a range of hardware. So, future upgrade of the MMI only requires minimum changes.

2. X provides only mechanism but not policy in designing an user interface. The "look and feel" of an application can be determined entirely by the programmer. By adopting a different Widget Set or Window Manager, the application can be made to emulate some other commonly available window systems.

3. Overlapping and resizeable windows in a hierarchy are important for presenting complex data by placing less burden on the user's short term memory.

4. High performance, high quality support for text, two dimensional graphics and images are useful in presenting graphical output.

However, X does have some drawbacks. For instance, X was not designed for animation. So, it is inadequate in producing fast and flicker free animation. The importance of animation and the method derived to solve the problem are presented in Chapter 6: The Interface. Nevertheless, the X Window System provides an excellent environment and versatile utilities to implement a powerful user interface.

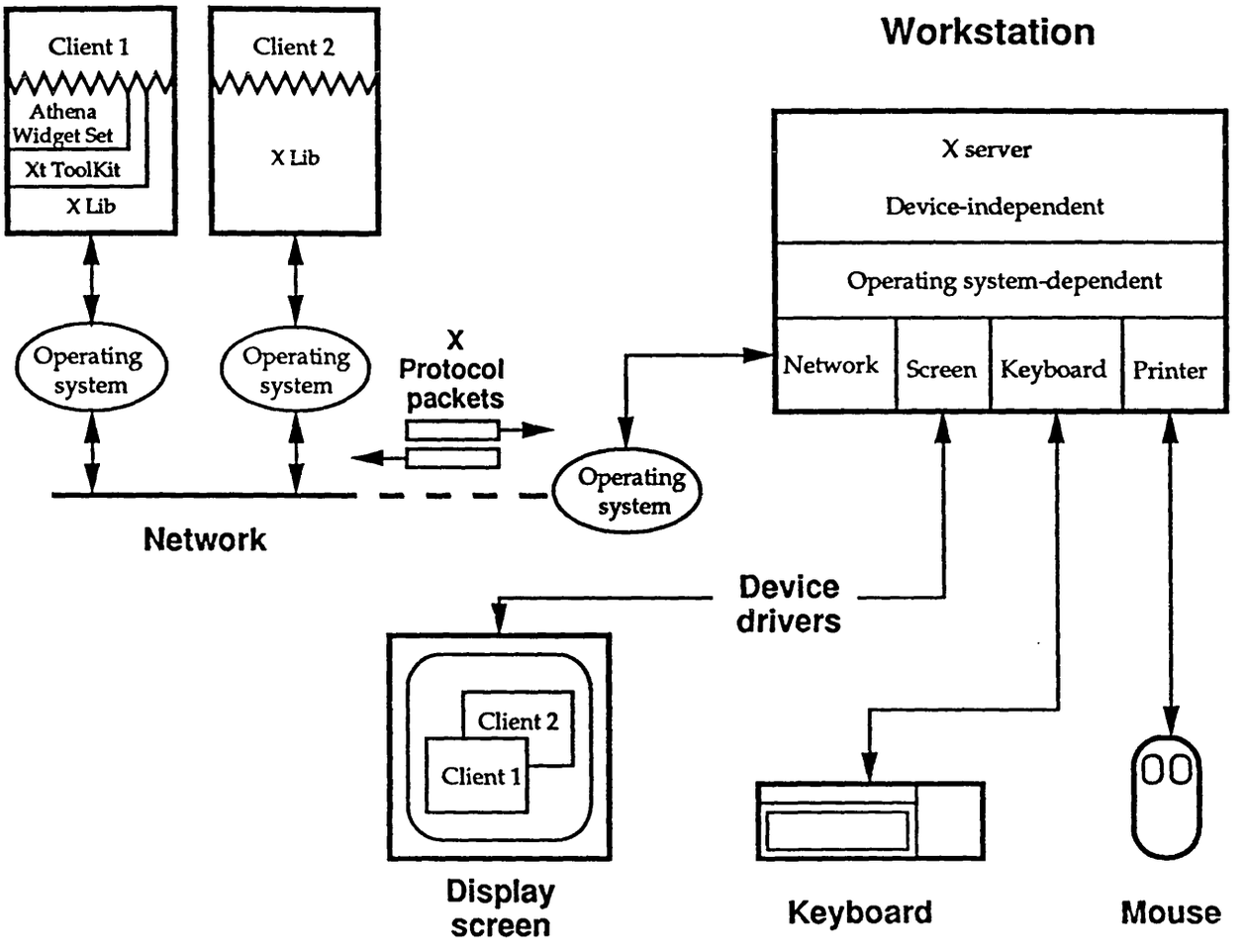


Figure 3.1: Structure of a typical X Window System

Chapter 4

Computer System Hardware

4.1 Introduction

Parallel processing has been used to implement the real time power system simulators at Bath University [2, 3, 71]. The simulation algorithms can be broken down into a number of tasks which can be executed simultaneously on a number of processing nodes. There are two advantages of using a multiprocessor system: Firstly, the computing power can be scaled up by adding more processing boards to the computer system without major modification to the simulation program. Secondly, because of the inherent parallel nature of the simulation problem, a relatively low cost multiprocessor system can be used to achieve satisfactory results which might otherwise only be feasible on a very expensive single processor computer.

The computer hardware used in the implementation of the MMI project is a locally designed multi-transputer system. A nineteen inch rack is used to host a number of processing boards. A maximum of sixteen boards can be accommodated. The connections of the boards are not hard wired. Instead, the boards

can be configured by the user in software.

The present five transputer rack is made up of the following processing boards:

1. Three T800 processing boards to provide the main computing power for the system.
2. One link topology configuration board to configure the topology of all the boards in the rack.
3. One I/O board to provide I/O services for the system.
4. One graphics board to provide high resolution bit-mapped graphics and to receive inputs from a mouse and a keyboard.

Interprocessor communications among the boards can be achieved in two ways: The first exploits the high speed transputer links [9]. The second makes use of a locally designed backplane [72]. While the former is adequate for the requirements of most parallel applications, the latter provides a fast and effective means for massive data transfer.

Fig. 4.1 shows the architecture of the multiprocessor system.

This chapter presents the functionalities, structures and operations of the processing boards. In addition, the backplane system for interprocessor communication is discussed.

4.2 The T800 Processing Boards

4.2.1 Main Structure

The T800 processing boards are the main processing units of the multiprocessor transputer system and run the Helios Operating System [8]. Each board is made up of the following components:

1. One Inmos T800 transputer [9].
2. One mega byte of DRAM.
3. Multiprocessor bus interface.
4. Local and multiprocessor bus arbiter logic.
5. High speed line drivers to connect the transputer links to the other boards via the backplane.

Fig. 4.2 shows the block diagram of a T800 processing board.

The T800 transputer is a VLSI single chip computer. It is made up of a RISC cpu which has a limited number of registers and simple machine instructions, 4 KBytes on chip fast SRAM, and four high speed DMA controlled serial links. The links have a maximum operating speed of 20 Mbits per second and are used for fast interprocessor communications. The address and data signals are multiplexed on its 32 bit memory bus. A built in memory controller is used to provide DRAM control and refresh timing.

Fig. 4.3 shows the internal functional block diagram of a T800 transputer.

The internal processor speed is link selectable and is generated by a 5 MHz external clock. In this way, the the CPU clock speed can be scaled up to 25 MHz with the peripheral circuits operating at 5 MHz, thereby avoiding the conventional problems of driving a circuit running at a high clock speed.

The T800 transputer can be booted from either one of its four communication links or from a ROM. The Helios Operating System is loaded up by a Tripos [73] shell script file which brings up a I/O server from the Tripos environment. The server then initiates the processing boards by sending bootstrap instructions to them via the links.

4.2.2 Interprocessor Communications

Interprocessor communication between the T800 processing boards can be achieved via the transputer links. Alternatively, the boards can communicate via the locally implemented backplane which is described in the next section. Essentially, the 32 bit transputer bus is connected to the shared memory bus via a bus transceiver. In other words, each processing board can access the local memory of another board via the backplane and regard all the available memory as a pool of shared global memory.

Hence, there are three main types of memory access cycles:

1. Local memory accessed by the on-board T800.
2. Off-board memory accessed by the on-board T800.
3. Local memory accessed by an off-board T800.

A simple method was adopted to identify the address of each memory location in each processing board as seen by an off-board processor. A unique number is assigned to each multiprocessor transputer rack that houses the processing boards. This provides a unique page within the whole memory span of the transputers. Each processing node has its own unique identity number within the page. The most significant byte of the address location denotes the rack number and the third most significant nibble indicates the identity number of the processing board. As there are at most sixteen processing boards in the rack, all transputer memory locations can be uniquely identified.

4.2.3 Memory Map Arrangement

The memory map of a T800 begins at 00000000H and ends at ffffffffH, of which one MByte is used by the Helios Operating System. The T800 recognizes 80000000H as its base address and the rest of one Mbyte is paged onto the memory page starting at 80000000H and ending at 800fffffH. The top four Kbytes from 80000000H to 80000fffH is not known to the Helios Operating System and can safely be used outside the Helios environment. It contains various information for bootstrapping, house keeping services, and interprocessor communication semaphore.

4.3 Link Topology Configuration Board

The Link Topology Configuration Board, LTCB, is used to allow full connectivity to be achieved in the nineteen inch transputer rack with up to sixteen transputers. The transputers can be configured by software in any arbitrary topology that is best suitable for an application.

The LTCB is essentially a 64-way static transputer link crossbar. There are two types of crossbar: dynamic and static. The dynamic crossbar allows link configuration to occur during the execution of a program. Effectively, direct links are established between all of the transputers. The static crossbar only allows link configuration before a program is initiated. The links connected are effectively a "hard" connection. Hence, the operating speed does not have to be high. In addition, the data bandwidth of the data flowing through the transputer links can be kept to a maximum and the delays in the links are only due to buffering and switching hardware. As a result, it is cost effective to adopt a static crossbar in the implementation of the LTCB [74].

The IMS-C004, or C004 in short, crossbar is specifically developed by INMOS [75] for link topology configuration in a transputer system. It is a 32-link crossbar which has thirty-two inputs and thirty-two outputs. The I/O can be connected inside the C004 to form any combination of link paths. Hence, one C004 can fully interconnect eight transputers together (Fig. 4.4).

The switch link is running at ten or twenty Mbits per second and has a worst case delay of two bit-time. The link signals are resynchronized before output and hence can be cascaded to any depth to form bigger switches. Switching is achieved by thirty-two 32-to-1 multiplexors which are controlled by thirty-two 6-bit latches. Five bits select the input to be connected to the output and the sixth enables or disables the output.

A rack of sixteen transputers can be split into four groups of four. One C004 can produce full interconnectivity for any two groups. Hence, six C004s are used to connect all the groups together (Fig. 4.5).

The five transputer rack used in the MMI project is connected in such a way that maximum practical connectivity is achieved. Fig. 4.6 shows the interconnection of the transputers.

4.4 Input/Output Board

The I/O board was designed and built by Hafeez [76]. It is based on the Philips SCC68070 [77] microprocessor, and is connected to a number of external devices in order to provide I/O services for the multiprocessor transputer system. A bidirectional, two wire data line made up of an IMSC012 link adapter is used to link the I/O board to the multiprocessor transputer system. Fig. 4.7 shows the block diagram of the I/O board.

The SCC68070 is a 16/32 bit microprocessor which has the same instruction set, programming model, internal mode as the more popular 68000 microprocessor to provide software compatibility. It has a 16 MByte addressing range and a 68000 compatible bus interface operating at 10MHz.

A on chip Memory Management Unit (MMU) is used to support virtual memory and to provide segment protection against illegal access.

A two channel Direct Memory Access (DMA) Controller is available to provide fast data transfer. DMA channel one is connected to a Shugart Associates System Interface (SASI) parallel bus which has a data transfer rate of approximately 4Mbytes per second. A 60Mbytes hard disk is connected to the interface to provide mass storage while a tape streamer is used to back up the hard disk. DMA channel two of the SCC68070 is used by the floppy disk controller DP8474

to transfer data to and from a 800Kbytes floppy disk drive which reads and writes files in Tripos format [73].

For interrupts generated by external devices, two programmable auto-vectors are available to register software routines to service them. This saves the CPU from having to poll the devices constantly. In hardware, a serial interface is used to connect the SCC68070 to the external devices.

The SCC68070 uses an Inter-integrated circuit (I^2C) bus interface to connect it to a real time clock PCF8583 and an I/O expander PCF8574, which can be used to provide a parallel centronic interface for a printer. The (I^2C) bus interface is a two wire, bi-directional serial bus operating independently of the centralized bus arbiter.

4.5 Backplane

There are two methods that the T800 processing boards can use to communicate with each other. The first uses a 32bit 80MBit per second multiprocessor bus, the backplane, shared by the processing boards. The second exploits the link bus for the T800 links. The first method was specially developed at Bath University to provide a powerful means for interprocessor communication in a multiprocessor environment.

The backplane runs at a clock speed of 20MHz and there are about 500 nanosecond interval between the address and data phases [72]. During that time, around ten other bus accesses can be started and/or finished on the same bus signal wires.

The backplane does not use any asynchronous handshake, address or cycle validation signals and is not customized for any particular processor. Each T800 processing board is connected to the backplane via I/O latching address and data buffers. When a piece of information is put on the bus during an interprocessor write cycle, it will only stay on for a short period of time. This allows the information to be latched by all the T800 processing boards which are ready to latch it at the time. On the other hand, when an interprocessor read is initiated by a T800, the data return is allowed as a separate cycle, which can occur at an indeterminate interval after initialization of the cycle. As a result, a variable access time for read cycles is achieved.

As the bus access is interleaved, it is possible that many bus accesses are made to the same processor. The problem can be solved by having two signals, SUCCESS and FAIL. When a cycle appears on the backplane, all processors decode the address of it. However, only one of them responds. If a processor has latched the information and carried out the cycle, the SUCCESS signal is then set by it. Otherwise, the FAIL signal is set instead. A failure leads to an automatic resend of the cycle two hundred and fifty nanosecond later by the initiating processor.

4.6 Bus Arbitration

In order to achieve optimum arbitration speed (i.e. to decide which T800 gets the next access to the bus), the arbitration mechanism has to be as fast as the multiprocessor bus access mechanism. Instead of using a standard centralized bus arbiter, a specialized distributed arbiter was developed. For processors with difficulty in getting time on the bus, limited priority shifting is provided.

The four least significant bits of the arbiter control signals are used for the fixed priority of a processor, which is set by the geographical position of the processor in the nineteen inch transputer rack. The most significant bits are used to change the processor priority under certain conditions. They are:

1. DATA - which makes sure that the priorities of the returning data phase accesses are always high as a processor is waiting to receive the data.

2. FAIR - which indicates that a processor has failed for more than sixteen time slots for trying to access the bus.

3. BDCST - which makes sure the rapid completion of the broadcast cycle. A broadcast cycle is one which is used by a processor to broadcast data to all other processors connected to the same backplane. Hence, the cycle is always of high priority. When a cycle is initiated by a T800, all other processors decode the present write cycle and pretend that it was meant for them. The SUCCESS and FAIL signals are then returned as before. If there is any FAIL signal, the broadcasting processor will then resend the cycle. At the same time, the processors which succeed in capturing the cycle will ignore subsequent broadcast cycles until a complete cycle occurs.

4.7 Graphics Board

The graphics board is responsible for providing graphics output, handling mouse and keyboard devices and serving as a T800 processing node. The board enables the five transputer rack to function as a basic X Window System workstation [78]. It is made up of the following devices:

1. One Microcore board [79].

2. One T800 transputer with 4 Mbytes of DRAM.

3. An INMOS G178 colour palette.

4. An external mouse and a keyboard connected to the graphics board via the Microcore board.

Fig. 4.8 shows the arrangement of the graphics board.

4.7.1 Microcore Board

The Microcore Board was developed by Philips Components Ltd. as an evaluation circuit. It is made up of the following main components:

1. A SCC68070 [77] microprocessor running at 9.83 MHz to handle an external mouse and a keyboard.

2. A VSC [10] to output high resolution bit-mapped graphics.

3. One Mbyte of display memory to hold the bit-mapped graphics.

4. A 64 Kbytes EPROM to hold an assembler program, MouseKey, which is responsible for setting up the VSC registers, polling mouse inputs and servicing keyboard interrupts. The program is listed in Appendix 6.

Video System Controller (VSC)

The VSC is a VLSI device integrating a 68000 family system controller and a bit-mapped colour graphic display controller. It offers full bit-mapped orgainization and can directly drive up to 2 Mbytes of RAM. The on-chip DRAM controller can support up to 1.5 Mbytes of DRAM and controls access to the unspecialised system of video DRAM. System ROM and peripherals are accessed by chipselect signals. In addition, a coprocessor interface is provided to allow very high speed memory access and manipulation.

As well as providing a variety of bit-map graphics manipulation, e.g. mosaic graphic effects, the CPU can access any memory locations even during active video display lines. Hence, system performance is greatly boosted.

The VSC is programmable via its registers and the local system is set up to have a display resolution of 720x480 at four bits per pixel, operating in interlaced mode.

4.7.2 T800

The T800 functions in a manner similar to that of the transputers on the other T800 processing boards described in the previous section. In addition, it is also responsible for writing bit-mapped graphics to the VSC for display and receiving mouse and keyboard data which are used by the X Window System [64].

There are two banks of memory: One bank has 4 Mbytes. The T800 memory is paged into two hundred and fifty six pages each of which is 16 Mbytes. The 4 Mbytes lie within the local page which starts at 80H and is used for running the

Helios operating system [8]. This bank is refreshed and maintained by the T800. The second bank of 1 MByte is used as display memory which is refreshed by the VSC. The T800 can access this bank via handshake on the VSC's coprocessor interface. To the T800, this display memory starts at 00H. In addition, the 68070 also places the mouse and keyboard data into specific memory locations within the display memory for the T800 to read and write.

Fig.4.9 shows the memory arrangement.

4.7.3 G178 Colour Palette

The colour palette, INMOS G178 [80], is used for flexible RGB output by fine tuning the intensities of the three primary colours: red, green and blue. The palette contains a colour lookup table in RAM which appears in the T800's logical memory page 01H. Hence, the T800 can read and write into the table. As four bits per pixel is used for the VSC, only a maximum of sixteen pixel values can be valid at a time. In other words, only sixteen colours can be seen on the screen simultaneously. This is adequate in displaying the simulation data.

4.7.4 Mouse

The Logitech Series Two mouse [81] is a programmable serial pointing device with three buttons. The assembler program, MouseKey, is used to handle mouse inputs activated by the user. When initiated, MouseKey enters an infinite loop and polls for mouse inputs. When the mouse is activated by the user, it will transmit a protocol of messages to the 68070 via the UART at the baud rate of one thousand two hundred. When the 68070 has correctly received an event, the

data is transferred to a sixteen bit address and an eight bit flag is set. Three sixteen bit addresses are used to hold the x, y positions and button state of the mouse. All the data and flag locations are pre-agreed between the 68070 and the T800. The flag is reset by a server running on the T800 once the data is read and the whole process then repeats itself.

4.7.5 Keyboard

The keyboard is IBM AT [82] compatible. The assembler program, MouseKey, is used to gather data from the keyboard. Each key press generates a unique code and when the key is released, bit seven of that code is set. When there is a key event (either a key press or a key release), the keyboard generates an interrupt to the 68070. Level four autovector is used by the processor to handle the interrupt. The vector is then disabled to prevent the processor from further interrupt. When the serial data has been converted into parallel form and read by the 68070, it is placed into a sixteen bit address and an eight bit flag is set. The data address and the flag locations are pre-agreed between the 68070 and T800. The flag is reset by a server running on the T800 once the data is read. After reactivating the interrupt vector when data is written, the 68070 is ready to accept new keyboard inputs.

4.8 Conclusion

The structure of the computer hardware used in the project has been described in the above sections. The computer is built as a multiprocessor system with a number of transputer processing boards and some additional functional boards.

The T800 processing boards provide the main processing power for the system. The I/O board provides the I/O facilities. The graphics board provides high resolution graphics and is also responsible for handling a mouse and a keyboard. A link topology configuration board is used to enable the system to be configured in any arbitrary ways in software by the user. Interprocessor communication is achieved either by the high speed transputer links or by the backplane. The computer system is readily scalable and is highly cost effectively for parallel processing. Because of the parallel nature of power system simulation algorithms, the multiprocessor system provides the best solution for the problem.

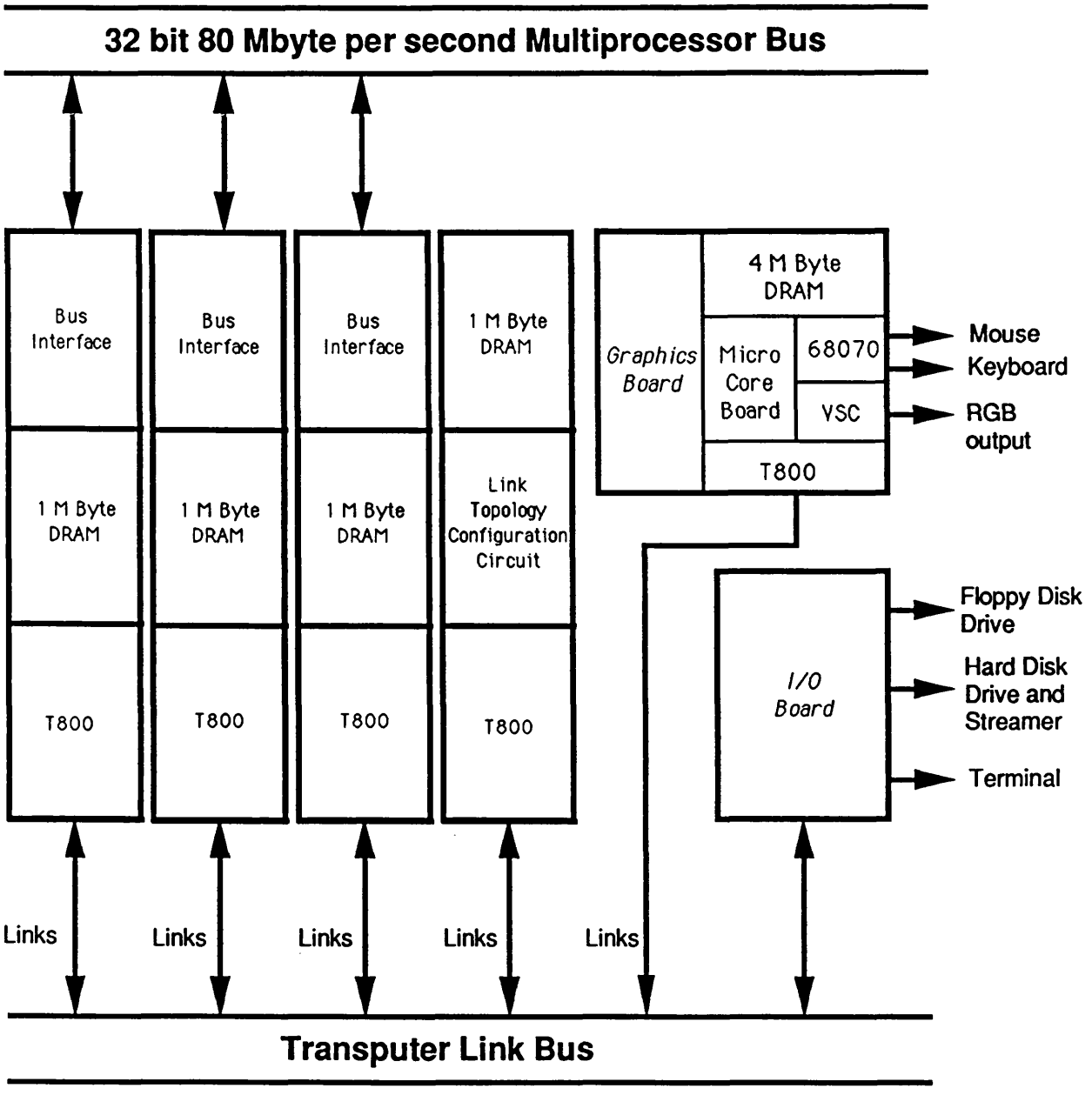


Figure 4.1: Architecture of the multiprocessor system

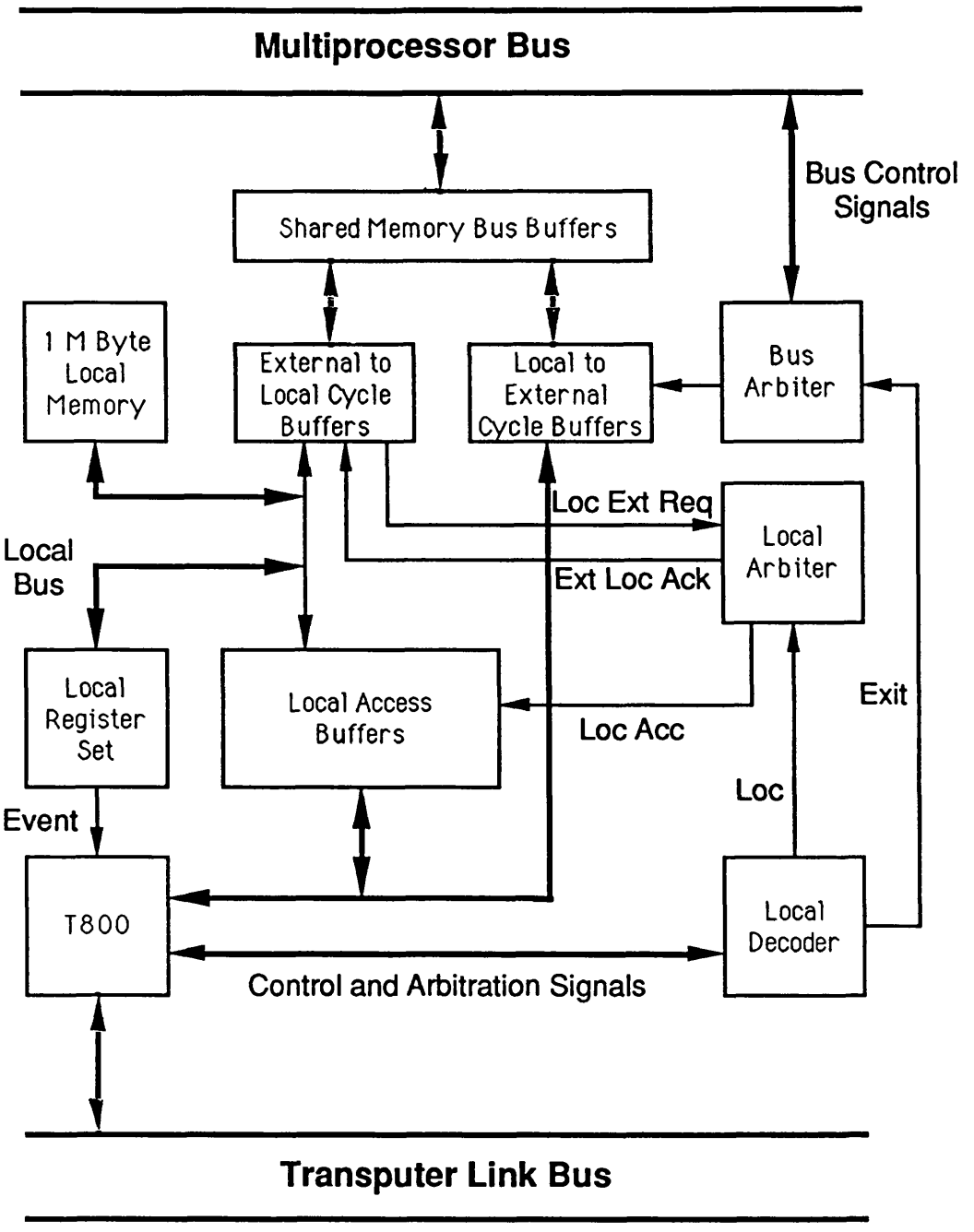


Figure 4.2: Block diagram of a T800 processing board

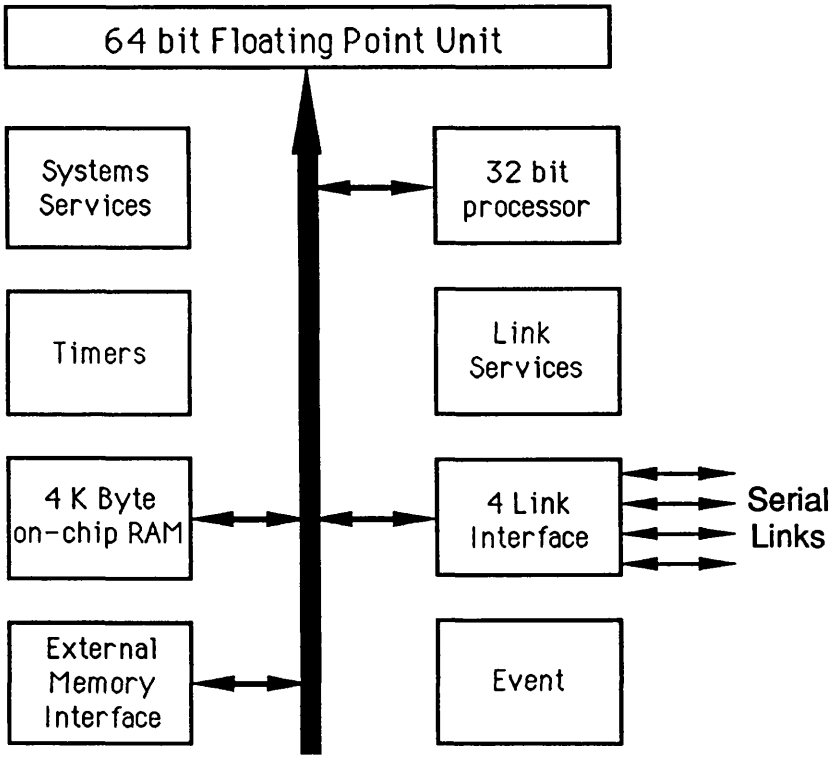


Figure 4.3: Internal functional block diagram of a T800 transputer

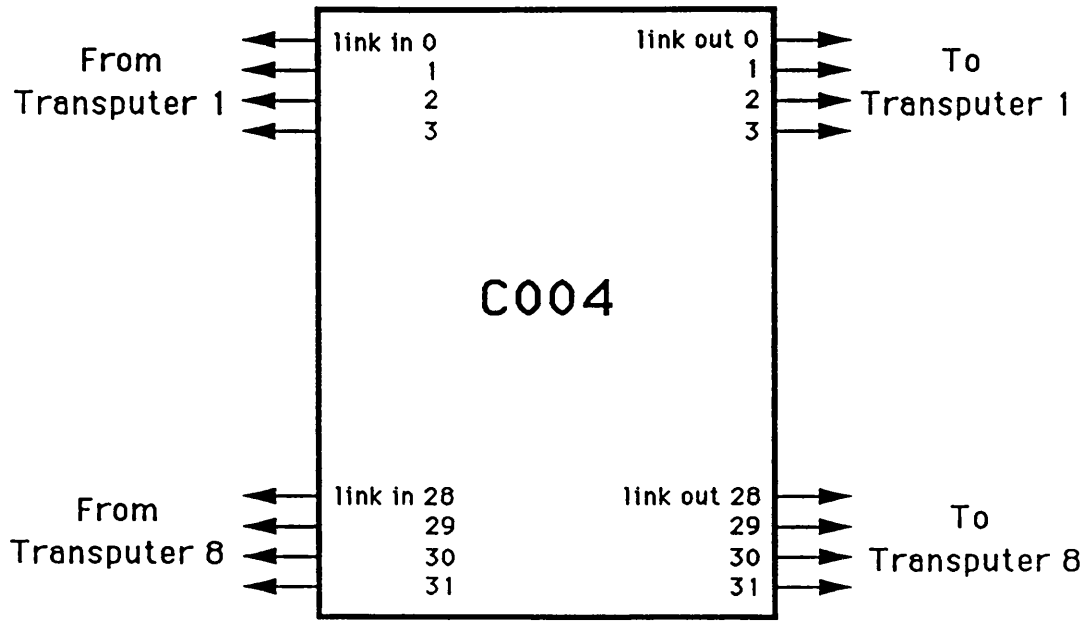


Figure 4.4: One C004 interconnecting eight transputers together

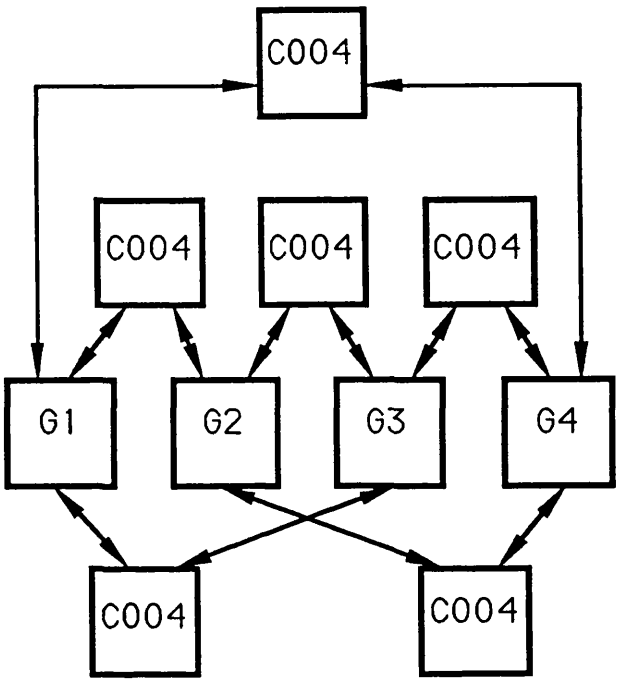


Figure 4.5: Six C004s interconnecting four groups of transputers

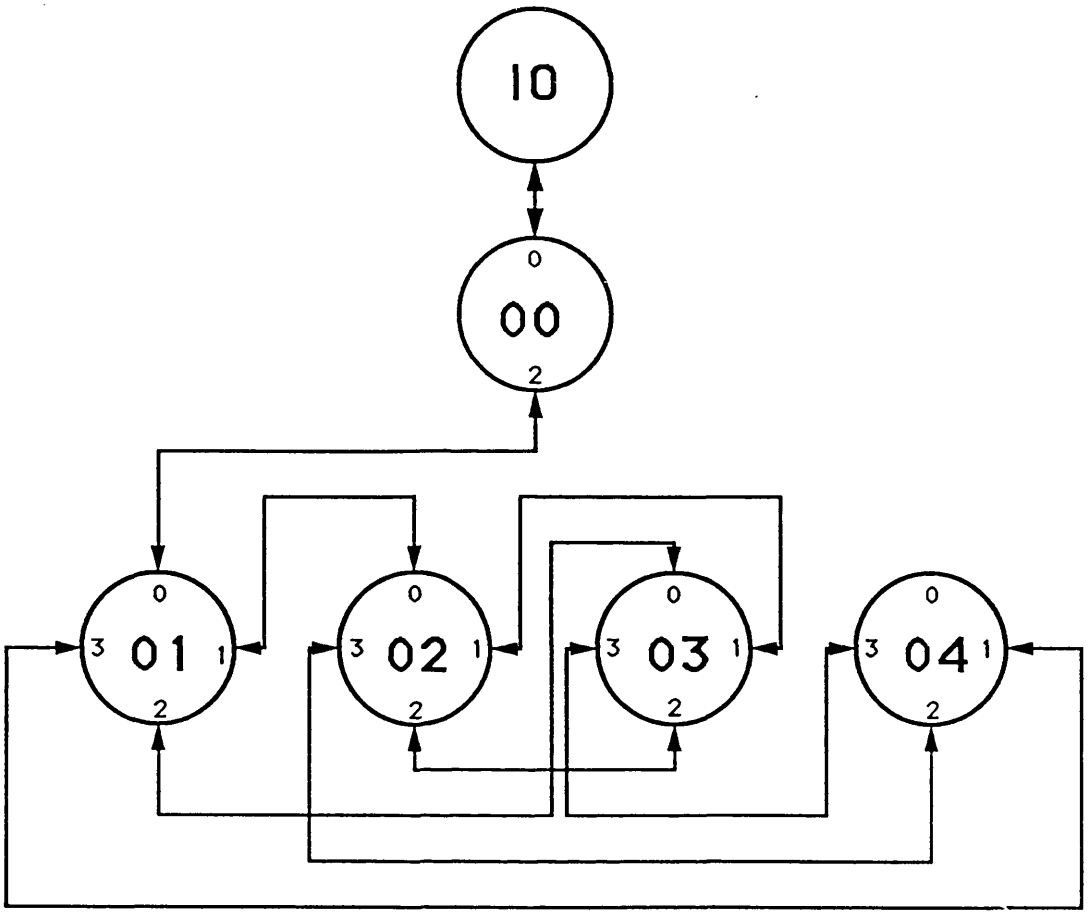


Figure 4.6: Interconnection of the five transputer rack

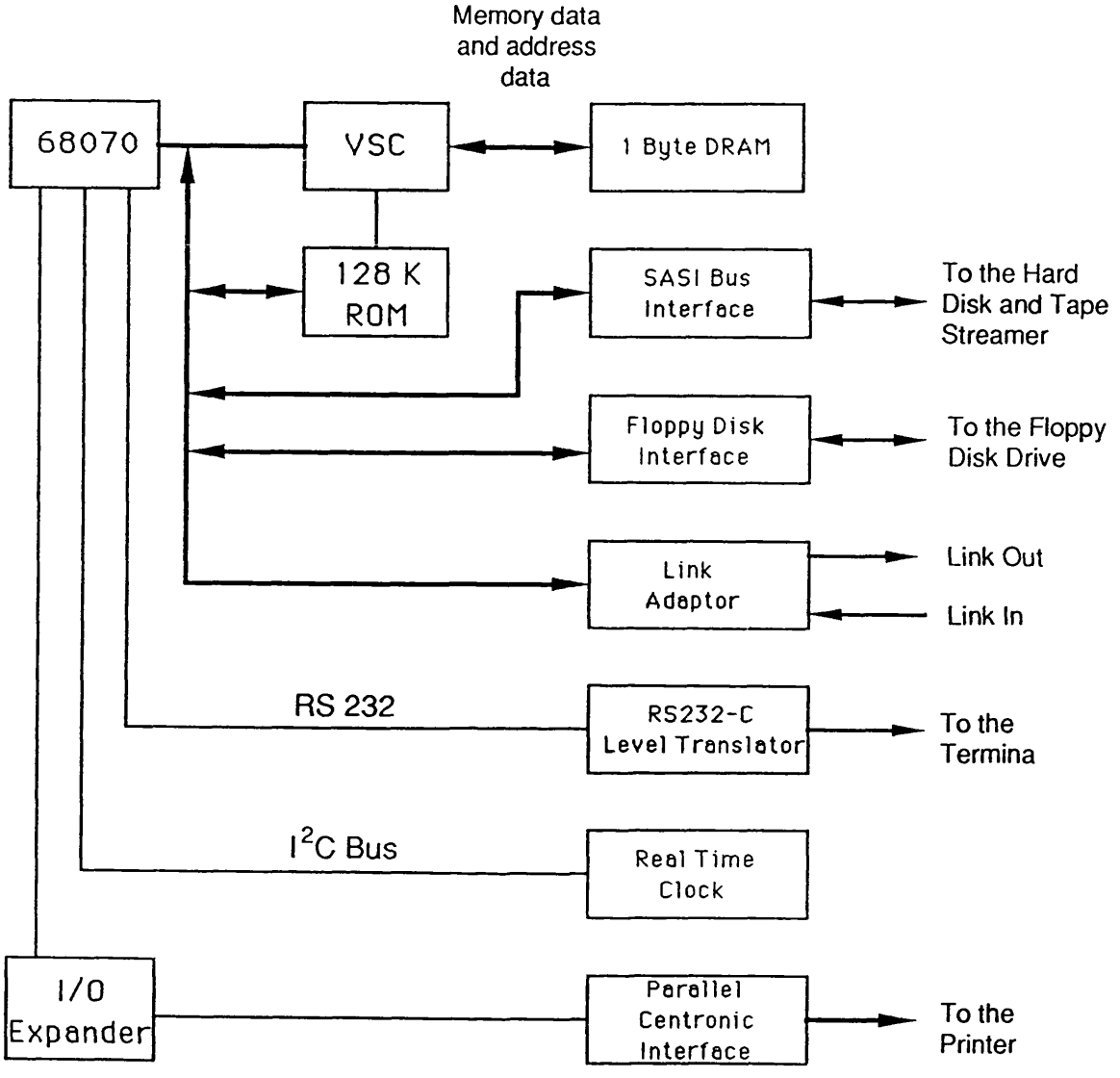


Figure 4.7: Block diagram of the I/O board

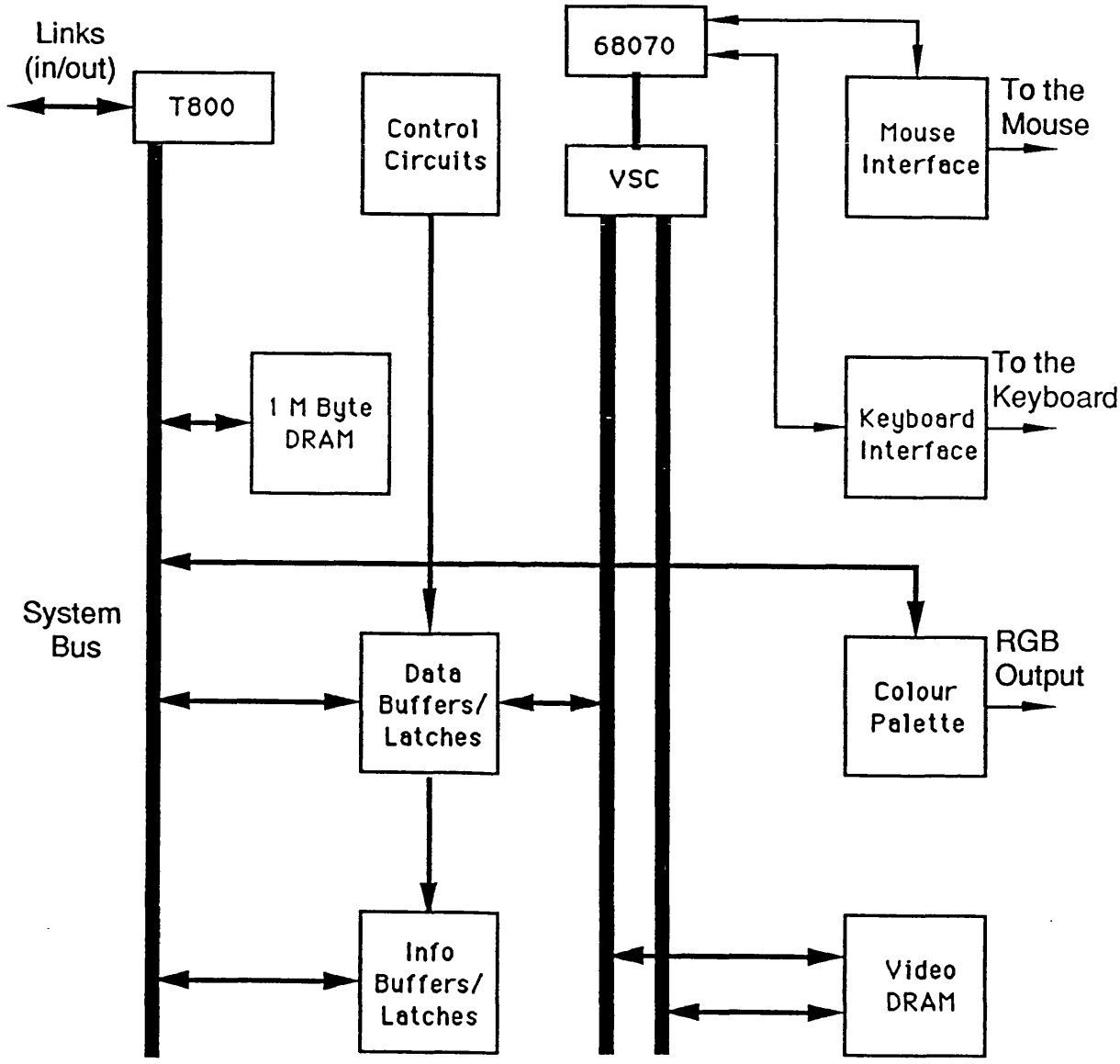


Figure 4.8: Arrangement of the graphics board

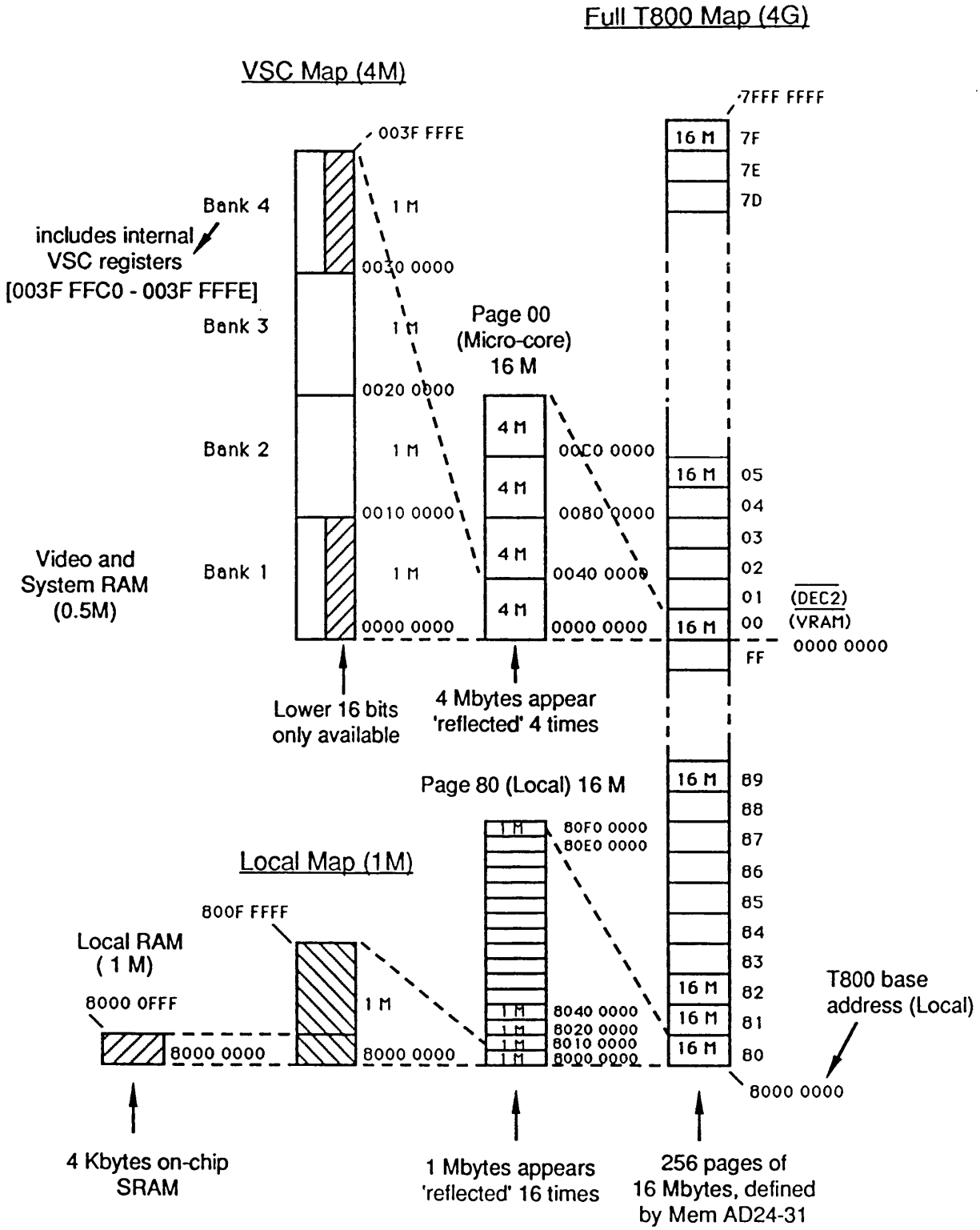


Figure 4.9: Graphics board memory arrangement between the T800 and VSC

Chapter 5

The Helios Operating System

5.1 Introduction

The choice of an operating system is relatively limited once the choice for the underlying computer hardware has been decided. The Bath mulitprocessor system based on the Inmos T800 transputers [1] naturally leads to two choices of operating systems:

1. Transputer Developement System (TDS).
2. Helios Operating System [8].

TDS is implemented in Occam [83] which is the native language of transputers. However, Occam is not very popular among other processors other than transputers. This seriously undermines the portability of the Simulator and a total rewrite of the software might be needed in the future. Hence, TDS was not considered as a viable option.

[1] Described in Chapter "Computer System Hardware"

Helios was developed by a software company, Perihelion Software Limited, which has a close working relationship with Bath University. The earlier version of the Simulator was developed under the Tripos operating system [73] which some members in that company were involved with. Helios is in itself a powerful operating system with advanced features as expected from a modern operating system. The native language of the system is "C" which is a very popular high level language [84]. Programs developed under Helios are then portable across a spectrum of computer hardware and operating systems [2]. Hence, it is only natural to adopt Helios as the new operating system to implement the next version of the Simulator.

The success of Helios among transputer users can be attributed to its advanced features and the facilities it provides to help to develop parallel programs. This chapter examines the design philosophy, structures and features of the operating system in detail.

5.2 Overview

5.2.1 Design Philosophy

Helios is a distributed operating system designed for multiuser, multiprocessor systems. Although it was originally developed for transputers based multiprocessor computers, it has been ported to a range of other computer hardware [8]. The hardware can be a powerful single processor workstation or a network of processors. SUN [85] is an example of the former while the Bath

[2] A version of the Simulator has been ported to the Apollo workstation running the Domain operating system

multiprocessor system [3] is an example of the latter.

Essentially, a Helios system is made up of a group of processing nodes, communicating and collaborating via the interconnecting network. Local processor groups are collected together to form a larger, hierarchical network. Auxiliary devices such as disk drives and printers can be connected to any clusters of processors and shared by every processing node in the network. Processors within a Helios system are not tied down to one user. Instead, they can be allocated dynamically so that the system can be expanded and modified easily with a minimum loss in performance.

Helios provides a consistent and uniform mechanism for accessing resources. For instance, identical system calls, such as Create, Locate and Open, can be used to manipulate processors, files and programs. This is achieved by writing system tasks that adopt a consistent server interface and obey a set of well defined protocol, the General Server Protocol.

To promote hardware portability, Helios hides the distributed nature of the computer architecture from the user. This layer of transparency enables a user to run parallel programs without prior knowledge of the processor network. Thus, carefully constructed programs can run across a platform of hardware without any software modification.

[3] Described in Chapter: "Computer Hardware"

5.2.2 Structure

Helios is made up of two major layers: The inner layer, Nucleus, runs on each node of the network and supports resource, communication and task management. The outer layer, User Layer, is made up of system servers and user programs.

The Nucleus consists of a few libraries and tasks. Its core is the Kernel which supplies the most elementary functions to the operating system. The Kernel is made up of the library, KernelLib, and some independent system processes. As a Nucleus must be present in every processing node, it is kept reasonably small. All higher level supports are implemented by servers and run on the User Layer.

The User Layer is also made up of a few libraries to support an environment to run C programs [84]. Servers are available to provide some operating system level facilities to access the underneath computer resource. For instance, the Network Server (NS) and the Task Force Manager (TFM) are used to let a user utilize the entire network in a transparent way. A user interface, Shell, is situated near the top of the User Layer. This is responsible for interpreting ASCII string commands and providing an Unix like environment for accessing the operating system [63]. User tasks are located right at the top of the User Layer.

In practice, Helios is built on a number of interacting tasks and a collection of libraries to support them. These tasks and libraries are distributed among the Nucleus and the User Layer. Details are described in the following sections.

Client/Server Model

Depending on their functionalities, tasks are categorized as clients and servers. Servers are responsible for controlling the hardware resources and clients are used to interact with servers to provide services to higher level tasks. A set of well defined protocols, GSP, is used to regulate the communications between clients and servers. Each server has a Dispatcher process, which waits for requests on its server port, and several processes to actually carry out the services required.

The GSP allows servers to be stateless so that they are not affected by crashes and communication losses. This makes Helios more robust in a multiprocessor environment. If a request is made to access an object, it follows the GSP. Fig. 5.1. shows the control vector of a request. The first three fields of the structure give offsets into the data vector which contains ASCII strings that define the object and its context. This capability is used by a server to check if a client has authority to access the context object. If a client has no capability, then no context name is given and the target name is given in null.

Some examples of the Helios servers are the Processor Manager (ProcMan) and the Loader supported at the Nucleus layer. Both are described in more detail in Section: "Servers".

Libraries

The libraries provide procedural interfaces to the operating system. The run time libraries are situated at the highest level. The Posix library is used to provide Unix compatibility so that the user can port existing programs from a Unix to

a Helios environment. Thus, a wealth of software is opened up as Unix is a well established operating system, making Helios more acceptable to new users.

As most of Helios is implemented in C, a CLib is provided to support standard C programs. The floating point calculations are supported in FpCLib. The library, FpLib is a processor dependent floating point library used by FpCLib. However, a user can choose to write customized libraries and link them in place of the standard libraries.

A general interface to the operating system is provided by the SysLib which is described in the Section: "Nucleus". The server end of the GSP protocol is supported by a set of procedures in ServLib. UtiLib has various procedures such as string manipulation and debug supports. The Kernel has its own library, KernelLib, which implements the lowest level operating system procedures and it is described in Section: "Kernel". As the Kernel has to deal with the hardware directly, some of the routines in KernelLib are written in assembly language.

Naming

Every item in a Helios network is regarded as an object. A naming system is used to uniquely identify each object and the advantages in this approach are [8]:

1. Any objects, e.g. tasks or user files, can be accessed in a consistent manner.
2. No specific knowledge about the network topology is required from the user.
3. With sufficient authority, all network services are available to the user.
4. Network extensions and modifications are dynamically accommodated.

In order to ensure that each object is unique, an unified naming scheme is implemented by Helios. A local name table is maintained by the Nucleus in each processor so that all the objects in the processor are contained. The tables are arranged hierarchically, just like a conventional hierarchical file system. Like the Unix operating system, files are grouped together into directories and directories are grouped together to form higher level directories. The root directory is situated right at the top. In Helios, the Unix concept is extended to include all the objects in the network. Each object is uniquely identified by its position in the hierarchy. A 32-bit descriptor is associated with each object name in the name table so that messages can be relayed to a server port and be serviced.

An example of a hierarchical network is shown in Fig. 5.2 The network is made up of two subnetworks, group A and group B. In order to merge the two subnetworks together, a higher node is inserted as the parent of group A and group B. The original names of the two groups need only minor changes. For example, the printer in group A was named "/GroupA/02/printer". In the merged network, the new name is then "/SiteA/GroupA/02/printer". There can be more than one object with the name "printer" and each one can be uniquely identified as the objects are distributed in different positions in the hierarchy.

5.3 The Nucleus

The nucleus is the core of the Helios operating system and it is copied to every processor in the network. It is the minimum system that must be present in every processing node. The main purpose of the nucleus is to control the resources of a single processor and to integrate them into the global network. As the nucleus is present in every processing node, it is kept reasonably small for efficiency

purposes. Hence, it only implements the most basic system support. Higher level system services such as file servers, resource managers and distribution of parallel tasks are implemented as high level servers which are added onto the nucleus.

The nucleus is made up of the following main components:

1. Kernel
2. System Libraries (SysLib)
3. Loader
4. Processor Manager (ProcMan)
5. Input Output Controller (IOC)

The structures and functions of these components are described in the following sections.

5.3.1 The Kernel

The Kernel is directly responsible for managing the hardware resources of a processing node. It contains KernelLib, which is a library of low-level procedures, and a few independent system processes, e.g. RAM disc server, FIFO server, to provide the following services:

1. Memory Management
2. Semaphores Management

3. Communication Primitives Support

4. Task Management

5. Event Handling

6. List Management

Memory Management

A processing node's memory is made up of a number of blocks which are collected together and organized into larger chunks called pools. When the system is booted up, all the free memory is collected into a pool, called the FreePool. When a task is created, the Kernel allocates a small pool from the FreePool and gives it to the task for its own use. When the task is finished, the Kernel combines the task pool with any other free memory blocks which are contiguous to it and puts them back to the FreePool.

Semaphores Management

In a multiprocessor environment, it is important to have a mechanism to protect critical data and to synchronize processes. A semaphore is often used for such purposes. Basically, a semaphore is just a counter which allows access to it by a single process one at a time. The process of reading or writing to the semaphore is performed in a single indivisible operation. The Kernel provides two routines, Wait and Signal, to increment and decrement a semaphore counter respectively. If the semaphore counter is decremented to a value smaller than zero by a Wait

operation, the task is suspended. Similarly, if the semaphore counter is increased to a value greater than zero, the task is restarted.

Communication Primitives Support

Helios uses message passing as the main means of communication and it is the lowest level of communication implemented by the Kernel. The communication provided is asynchronous, "unreliable", blocking and point to point. Error checking or re-transmission are not supported at the Kernel level. Instead, they are the responsibilities of higher level procedures.

There are two primitives for transmitting and receiving data: PutMsg sends a message to a port, which is a software entity identified by a 32-bit descriptor, PORT, in the port table. GetMsg receives a message from a local port. There are two types of message passing: The first only involves processes and ports on the local processor. The second might involve a number of intermediate ports for relaying a message from a local port to a remote port on another processor. The detail of communication is explained in more depth in Section: "Communication Methods in Helios".

Task Management

The Kernel provides procedures for initiating and killing tasks. The tasks are created and manipulated in the Nucleus level by the ProcMan and do not run at the Kernel level.

In order to create a task, the processor loads the program image of the task in to the memory, and the Kernel routine InitTask is called. This routine allocates the stack, heap memory and the static data from the image header and the module header respectively. The module initialization code is used to initialize the static data. A module table is used to allow sharing of code segments by different tasks by containing an entry associated with each module.

The Kernel routine, KillTask, is used to terminate all the processes belonging to a given task.

Event Handling

The Event signal line is the only external hardware input on the transputer other than the four links. This acts like an interrupt and is shared between many devices as there is only one event signal.

The Kernel routines, SetEvent, is responsible for installing an event handler routine. A high priority kernel process waits for events to be signalled. When an event is raised, the event list is scanned and the relevant event handler is called in turn. To avoid event handlers from being suspended, direct calls to the message routines are prohibited. The kernel routine, RemEvent, is used to remove an event handler from the event list.

List Management

To provide efficient dynamic data management, the kernel data is arranged into a number of double-linked lists. As a by product, the routines used by the Kernel for internal data link-lists management are made available to the user.

A list is made up of nodes (Fig. 5.3). In practice, the head points to the first node and the tail points to the last node in the list. The earth node is used to make sure that none of the nodes in the list has a null pointer and avoids special cases in list handling routines. A node can be used to form the start of a larger data structure so that a number of these structures can be linked together.

Routines are provided to initiate a list, insert a node before or after a given node, remove a node, add or remove a node to the head or tail of a list, walk through a list and apply a function to each node, and walk through a list and search for a node which satisfies a certain criterion.

5.3.2 System Library (SysLib)

SysLib is a resident sharable library which provides a general interface to the operating system services. These services are similar to those provided by the Unix system calls [63]. To promote portability, this library is independent of any programming language. In fact, many SysLib procedures are only higher level interfaces to the underlying low level message interactions.

The main task of the SysLib is to interact with server processes. It directs requests to the IOC and operates a few small housekeeping functions such as monitoring

the system resources allocated to a task and releasing them to the system pool when the task is finished.

5.3.3 Loader

The Loader is an autonomous process responsible for loading programs and data structures into a processor's memory. A directory interface is supported to examine and manipulate all loaded objects.

5.3.4 Processor Manager (ProcMan)

The ProcMan is an autonomous process responsible for managing the hardware resources of a processor. It prepares the underlying structure necessary to run a task before it is created. During the lifetime of a task, it is managed by the ProcMan. When it dies, the ProcMan will tidyup all the resources used by the task. A standard directory interface is provided by the ProcMan to examine and manipulate the running tasks. The ProcMan is also responsible for creating a name table for all the objects, e.g. tasks associated with the local processor. The table is used by the IOC to search for an object (more details in Section: "Naming" and Section: "IOC").

Special attention is required from the processor if an exception occurs, e.g. stack overflow, arithmetic overflow and console attention. A signal is sent to the ProcMan responsible for the task. The ProcMan then reports the exception to the program by invoking an exception routine provided either by the runtime system or by the program itself.

5.3.5 I/O Controller (IOC)

When a task is created, the ProcMan also spawns an internal IOC process for it. The process is responsible for handling I/O requests from the rest of the network on that task.

For every processor, there is a name table containing all the names of the objects associated with it. All IOCs on the processor can access the table. When a request is made to an IOC about an object, the Controller will try to locate the name of that object in the name table by interacting with a server, Name Server. The server is in fact just a child process spawned by the IOC when an access to the name table is needed. If the search is successful, the request is passed on to the server whose port is indicated in the table entry. Then, the object can be accessed by the server. However, if the object name is not present, the IOC will initiate a distributed search amongst the neighbouring processors until either the name is found or not. Sometimes, the search will propagate throughout the entire network. When the name is finally located, an entry for it is created in the name table saving any future search for the object.

An IOC process is also created for each link of a processor. The process is used to respond and propagate distributed search requests. It also functions as an agent for tasks in the remote processors when they need to access objects that are local to the IOC.

5.4 Communication Methods in Helios

There are four levels of communication methods available in Helios:

1. Language Level I/O. This is the highest communication level and is highly language dependent. A stream is opened for a file with a specified name which is later used as an identifier to allow data transfer to or from the file. The data recovery is high and hence the routines in this level are used to form a higher level interface for the data communication between two Helios objects.

2. Posix Level I/O. The routines in this level are used for data communication between two Helios objects. If a file is used to transfer data to or from it, a file descriptor is created to identify the underlying file stream. If the data is being read from or written to a task, then a pipe or a FIFO is associated with the file descriptor. An error recovery mechanism is also available.

3. System Level I/O. The routines here use Helios streams for data communication. Error recovery and data integrity checking are also available to safeguard communication. Although the overhead of these routines might be high, programs written with them are more portable and safer.

4. Kernel Level I/O. This is the lowest level of communication supported by Helios. They provide a means of fast communication and are supported in the Kernel level. More detail is shown in the next section.

The communication overheads in the case of a transmitter and a receiver residing on adjacent processors using the above communication methods can be summarized in the table below [86]:

Table 5.1: Communication performance using Helios primitives

Message size (bytes)	Posix level read/write (micro second)	System level Read/Write (micro second)	Kernel level GetMsg/PutMsg (micro second)
4	1110	1100	125

5.4.1 Kernel Level I/O

Helios Message Structure

A Helios message is made up of three parts: a header, a control vector and a data vector (Fig. 5.4). Both the destination and the reply ports of the message are specified. The maximum amount of data that can be contained in one message is 64 KByte. If the amount of data to be transfered is variable, the data size should be added into the DataSize field. As the message body is divided into control and data parts, the message can be transmitted from or received into different areas of memory. Either or both of these portions of the message might be missing.

Helios Ports

A port is a software entity that is created by the Kernel to make message passing transparent to the user. A port table is used to hold all the ports available to a processor and each port is identified by a 32-bit number, PORT (Fig. 5.5). "Index" is used as a word offset into the port table. "Cycle" is used to differentiate between the states of the port as it might be reused over and over again for passing different messages. "Uses" is used for garbage collecting ports which are unused as a program is crashed or failed to tidyup.

There are two types of ports: The first is a local port for messages communicating between processes on the same processor. The second is a surrogate port for relaying messages from a remote process to another one on other processor.

Communication Primitives

There are three message passing routines supported at the Kernel level:

1. NewPort().
2. PutMsg(MCB).
3. GetMsg(MCB).

When a program needs a new port, it calls NewPort to locate an unused slot in the port table. After initialization, the routine returns a descriptor for the port. If there is not a free slot in the table, the routine looks for the least recently used slot and reuses it. In order to maximize the time elapsed between a port being freed and its slot being used, the slots are allocated in a cyclic order.

Before a message can be sent, it must be initialized to a special format. For that purpose, InitMCB and a set of message marshaling routines are used to prepare a Message Control Block (MCB) for a piece of data (Fig. 5.6).

PutMsg first checks the validity of a port descriptor in the port table. If it is correct, the message header is transmitted on the port channel, followed by the control vector and then the data vector. As it takes a while to transmit a message, some processes might not be able to wait. Hence, a timeout is specified in the MCB header. Upon expiry of the timeout, the routine returns with an error code. A timeout of -1 makes the routine waits for an indefinite period of time.

GetMsg checks the port descriptor contained in the MsgHdr.Dest field and waits for a message header from the channel. It then receives the control vector and

data vector according to the values of MsgHdr.ContSize and MsgHdr.DataSize respectively. Similar to PutMsg, GetMsg also operates with a timeout and a timeout value of -1 causes the routine to wait indefinitely.

If a program can not afford to wait for the message transmission to finish or timeout, in the case of a failure, it should spawn a child process to perform the communication. The parent process should be allowed to attend to some other jobs. Moreover, a port can be shared by several processes but only one of them can be active at one time. Hence, a program which has to wait for messages on more than one port has to spawn a child process at each port.

Network Messages

If a message is destined for a port on another processor, it must be relayed over more than one port. The structure of such a relay port, Surrogate Port, is shown in Fig. 5.7. The mechanism for transferring such a message is as follow:

The Helios interprocessor communication is based on the transputer link structure. A Link structure is associated with each serial link. A high priority Kernel process, called Link Guardian (LG), is responsible for controlling each link. As messages have to pass through a hardware link to a neighboring processor, the LGs effectively control the network message traffic.

When a message is sent, PutMsg detects the types of the destination port. If the message is meant for a local port, everything behaves as described in the previous section. However, PutMsg behaves differently if the port is not a local one.

After being granted a link, PutMsg transmits the protocol header byte, MsgHdr structure, control and data vectors, if any are present. When the protocol header byte and MsgHdr header are intercepted by the LG, the reply port in the header is replaced by the descriptor of a newly allocated surrogate port in the port table. The surrogate port holds all the information needed to route a reply message to the originating processor from the destination port. LG then looks up the destination port in the port table and its type is examined. If the port is a local one and a receiver is waiting for the message, LG sends the header to the receiver and waits for the control and data vectors to be taken by the receiver via the link. Then LG resumes control. If there is no receiver present, the message is put into a buffer temporarily and queued for later delivery. If the destination port is a surrogate port and the next link is available, LG then transmits the protocol byte and message header. The control and data vectors are received simultaneously and are transmitted down the link when it is ready. If the link is busy, the message is stored into a buffer and queued for later transmission. The message is thrown away if no buffer is available.

When there is too much traffic, a message could be thrown away. Then, an exception message is generated and sent back to the reply port. Although the LG makes slightly more effort in delivering the exception messages, they might still be lost if the network traffic is too heavy.

5.5 Servers

The system servers, Network Server (NS) and Task Force manager (TFM), are fundamental to the network management of a distributed Helios system. The Host Server is valuable in providing I/O facilities to a Helios network. The three

servers are described in this section.

5.5.1 Network Server (NS)

The Network Server is used to initiate and control the Helios network. It is distributed hierarchically among the network with each of its separate component responsible for each subnetwork.

A text file contains a description of the physical configuration of the network, the type of processor in each processing node, any special hardware attached to a node and how each node is connected to the other nodes. A program, called rmgen, is used to generate a resource map from this file. The NS will configurate the Helios network according to this map.

5.5.2 Task Force Manager (TFM)

The Task Force Manager is a hierarchically distributed server. It is made up of a number of identical servers which are distributed throughout the Helios network. Each of these servers are responsible for controlling a TFM portion of the whole network.

The TFM maintains detailed information about the resources that it controls. It also gathers information about the relative loading of the subnetworks and the connectivities between them. The TFM has the capability of allocating jobs for each subnetwork and is responsible for load balancing. It processes all client level program execution requests and analyses the current state of the network. The component tasks are then distributed throughout the network according to the

following criteria: Optimal allocation of resources; polarization of different user task force; maximum efficiency of interprocessor communications; maximum parallelism. As Helios does not support dynamic load balancing, the best mapping must be achieved before the task force [4] is executed.

After a task force has been scheduled, the TFM then keeps track of each component task. When all components have been terminated, the client is informed.

5.5.3 Host Server

Helios can be bootstrapped from ROM, disk or a host machine which is used to provide I/O services to the Helios network, e.g. console and disk access. The Bath system uses a Tripos [73] system as the host machine. A host server runs under the Tripos operating system and is responsible for booting up Helios, emulating the host machine as a Helios node, providing I/O services and debugging support.

When booting up Helios, the Host Server will take the Helios Nucleus image from the disk and load it on the Boot Root Node via the communication link. Once the image has been successfully loaded, the processor will jump to the KStart function in the KernelLib and start up the Kernel. After that, the rest of Helios is loaded on that node. A LinkGuardian process will be started by the Kernel in each link and one of them will send back a message to the Host Server to indicate a successful boot.

Once Helios has been booted up, the Host Server is ready to receive messages from the rest of the network. The Server is made up of a main loop which waits for

[4] A task force is a collection of tasks

messages, and a few servers which deal with objects like the file system, console, etc. Messages will be forwarded to the appropriate server as they come in.

The Host Server has a server called IOProcessor which emulates the host node as a Helios node so that the node does not need special treatment.

As well as providing I/O services, the Host Server also supports a certain amount of debugging. This includes examining the processor memory and debugging program code. The Kernel trace vector, which is an array in the Kernel, is also interpreted by the Host Server so that system processes can read and write information in the vector for tracing program error.

5.6 Parallel Programming Support

There are two ways to run parallel programs under Helios:

1. using Component Distribution Language (CDL).
2. writing parallel algorithms with Helios libraries.

5.6.1 CDL

CDL is used for coarse grained parallel problems. In each case, there are several programs running separately and communications between them are via pipes. For example, the output of a compiler can be directed via a pipe to a linker when a program is being compiled. Thus, the whole process is speeded up.

There are two ways that CDL can be used: The user can either explicitly declares the components of each task in the task force, or write a CDL script to define the task force. The CDL syntax is an extended version of the Unix C Shell [63]. In addition to the Unix pipe constructor, four other parallel constructors are used: reverse pipe, simple parallel, subordinate and farm. A complex task force can be described precisely by using these constructors.

5.6.2 Parallel Algorithm

Although CDL is quite flexible in allowing user to specify how a task force can be constructed, it is rather inefficient in dealing with fine grained parallel problems. Either it is too tedious to write a detailed CDL script or the efficiency is not satisfactory.

Helios provides library routines for creating tasks and some other necessary programming constructs to enable the user to write parallel programs from scratch. The user can then make use of the intrinsic parallelism in the problem and tailor make a parallel solution for it. However, the disadvantage of this method is that the user has to be well acquainted with Helios.

5.7 Conclusion

The design philosophy, structures and features of the Helios operating system have been examined in this chapter. The reasons for the adoptation of the system can be identified as:

1. High portability of the programs developed under the operating system. As Helios offers Unix compatibility, the "C" programs developed under the system are readily portable.

2. Advanced and powerful operating system support for multiprocessor environment. Facilities such as task management and primitive communication supports are valuable in parallel processing. Due to the inherent parallel nature of the power system simulation problem, Helios in a multiprocessor environment offers the most cost effective solution.

```
struct    IOCCommon{
          Offset        Contex;
          Offset        Name;
          Offset        Next;
          Capability    Access;
};
```

Fig.5.1 Structure of a Helios Input Output Controller Data

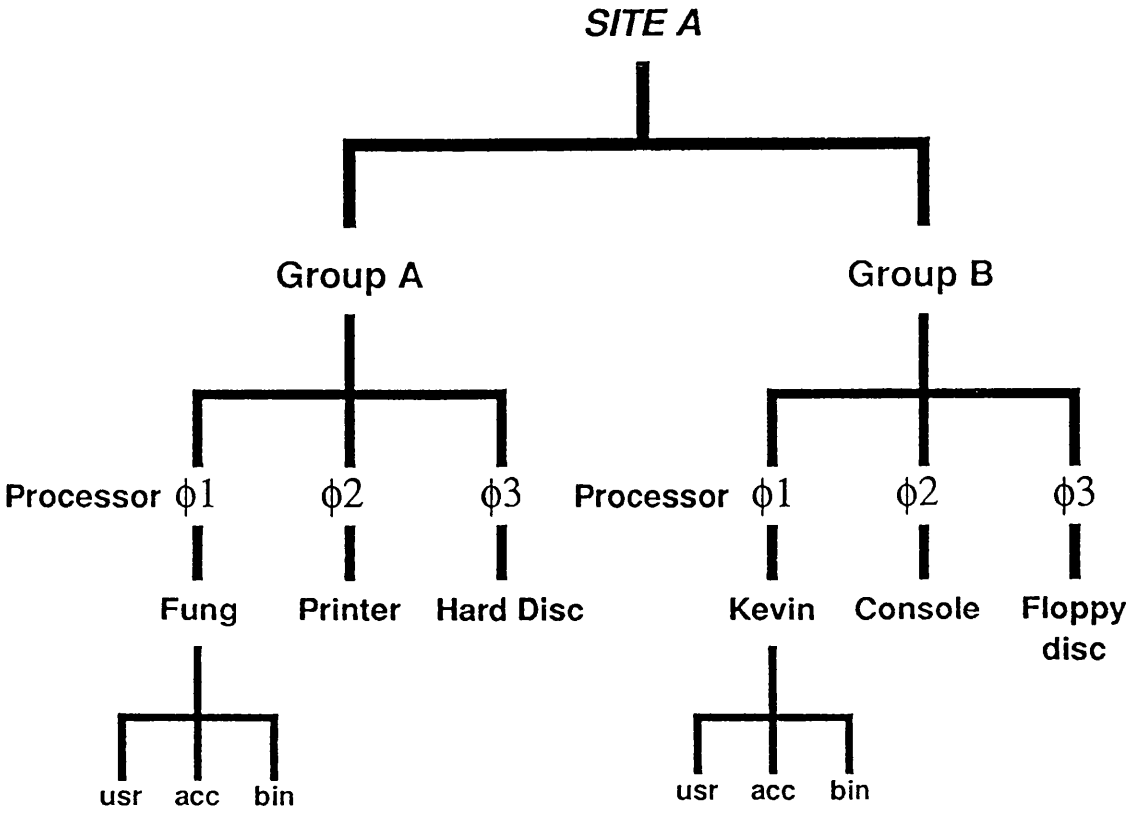


Figure 5.2: An example of a hierarchical Helios network

```
struct    Node    {
          struct  Node    *Next;
          struct  Node    *Prev;
};
```

Fig.5.3a Structure of a Helios Node

```
struct    List    {
          struct  Node    *Head;
          struct  Node    *Earth;
          struct  Node    *Tail;
};
```

Fig.5.3b Structure of a Helios List

```
struct    Message   {
          struct    MsgHdr    MsgHdr;
          word                Control[...];
          byte                Data[...];
};
```

Fig.5.4a Structure of a Helios Message

```
struct    MsgHdr    {
          unsigned  DataSize:  16;
          unsigned  DataSize:  16;
          unsigned  DataSize:  16;
          Port      Dest;
          Port      Reply;
          word      FnRc;
};
```

Fig.5.4b Structure of a Helios Message Header

```
typedef     struct      {
            INT16       Index;
            byte        Cycle;
            byte        Flags;
};          PORT
```

Fig.5.5a Structure of a Helios Message PORT descriptor

```
typedef     struct      Port        {
            byte        Type;
            byte        Cycle;
            bits        TxState:    2;
            bits        RxState:    2;
            bits        Flags:      4;
            byte        Uses;
            Channel     Chan;
            word        *TxId;
            word        *RxId;
};          Port
```

Fig.5.5b Structure of a Helios Message Port

```
typedef     struct      MCB{
            MsgHdr      MsgHdr;
            word        TimeOut;
            word        *Control;
            word        *Data;
};          MCB
```

Fig.5.6 Structure of a Helios Message Control Block

```
struct    IOCCommon{
          byte        Type;
          byte        Cycle;
          byte        Flags;
          byte        Uses;
          Port        Port;
          word        *TxId;
          word        UnUsed;
};
```

Fig.5.7 Structure of a Helios Surrogate Port

Chapter 6

The Interface

6.1 Introduction

Interactive power system simulators have been available for some time [87–103]. The analysis of power system performance can be enhanced by close interaction between the engineer and the computation. Recent development in high quality bit-mapped graphics displays and powerful microcomputers are encouraging the major revision of computing practice for power system analysis.

This chapter presents a survey of the previously developed power system simulators around the world. The uniqueness of the Bath system, and hence the problems associated with it, are also highlighted. Having discussed some user interface theories [1], X Window concepts [2], the host computers hardware [3] and the operating system [4], it is then possible to present the conceptual organization of the MMI. The software hierarchy, dialogue model, presentation style, interaction style, colour scheme and various design issues as raised in Chapter 2 are

[1] Refer to Chapter 2
[2] Refer to Chapter 3
[3] Refer to Chapter 4
[4] Refer to Chapter 5

discussed. Finally, the importance and problems encountered in implementing real time animation, and its effects on the user interface design are also addressed.

Throughout this chapter, emphasis is placed on power system simulators, and power system engineers are regarded as the end users.

6.2 Survey of Interactive Power System Simulators

6.2.1 History

The benefits of using an interactive system are: "to give the user an excellent qualitative 'feel' for the behaviour of the system, and to aid greatly in the selection of alternatives, anticipation of problems, and interpretation of results" [87]. This is in stark contrast to the batch-processing computers that power system engineers relied on in the early days of computing. Those systems might be fast for computation, they were very poor in user interface and the interaction with the user was always hampered by inadequate displays and low rates of data communication. The engineer felt isolated from the computation process and it had the following consequences [5] :

1. It took longer to make a closure in the mind as the computation turn around time was usually very long.

2. There was not a smooth skill/rule/knowledge transition process. A user was highly demotivated.

[5] Refer to Chapter 2 for the background information in the criticism

3. The communication bandwidth between the user and the program was very limited because of :

    - the computer input/output bottleneck, e.g. poor displays for simulation results.
    - the mental bottleneck, e.g. the data output to the user were usually in numerical form which was extremely hard to digest.

4. A user could not understand the computer program and hence his model of the system was inadequate and faulted, i.e. magical thinking might arise.

In conclusion, early day batch processing computer systems were very bad for user interface and led to poor performance of the power system engineers using such a system in analyzing a power system.

With the advent of high quality colour displays, powerful microprocessor technologies and related breakthroughs in computer hardware, it is now possible to build a high performance and interactive computer system. In particular, as the importance of Man Machine Interface has now been realized, the needs, behaviours and characteristics of human beings are much better understood. A system which a user actually wants and can use to accomplish his task, can now be developed to help in analyzing a power system, without being distracted and hindered by the tools used.

6.2.2 Characteristics of an Interactive System

Before proceeding any further, it is necessary to discuss the characteristics of an interactive system, so that a comparison of the various simulators that will

be presented can be made. The characteristics of an interactive system can be identified as :

1. allowing the engineer to make modifications as to how the simulator should be run and what subsequent actions should be taken, during the course of simulation.

2. maintaining a dialogue between the engineer and the computer. The communication between user and computer may be terminated and reopened at any time and either may interrupt the other at reasonable intervals without fear of offending.

3. not requiring the user to store large amounts of precalculated information. The computer should have the ability to quickly calculate any desired quantity so that the user can look at different data at will without excessive waiting.

6.2.3 Survey

Undril [87] addressed the importance of an interactive working environment and assessed the computer requirements as to how an interactive system could best be set up. The software structure and some man machine interface theories were also discussed. The system was mainly text based.

Alvarado [89] introduced several new concepts in user interface. A user could build up complex command sequences within a text file, called macros. Macros could be called iteratively during the program provided that they were pre-defined before program initialization. A user could custom-define faults without repro-

gramming, simply by changing a configurable text file. Software monitors could be placed anywhere within the power system network, as defined in a text file. They were used to examine the flow of current in detail. The system was also mainly text based.

With the introduction of high performance microcomputers, emphasis has been placed on utilizing graphics to display simulation data. The dialogue between user and computer was based on selecting a text menu or entering a command line via a keyboard [91–96, 100, 102, 103]. As WIMP systems became more popular, windowing systems with mouse pointers were used implementing the user interfaces of several simulators [98, 99, 101]. In particular, Fujiwara [90] used animation to display dynamically changing data, in order to give user a vivid picture of the system state. Chan [97] argued the inadequacy of text-based dialogue and adapted direct manipulation techniques to let a user issue commands and interact with the power system entities displayed on a one-line diagram. Some interesting user interface theories as applied in a power system simulator were also addressed.

6.2.4 The Bath System

Bath University has been developing power system simulators since early in the last decade [2,3]. As the complexity of the simulated power system grows, so does the user interface evolve at a rapid speed. The user commands and simulation results become ever more complicated. The user interface created by Dale [2] was command line based, with animation diagrams and graph plots used to display simulation results. Two monitors were used to enhance communication between the user and the computer. A colour monitor was used to display the power system and a monochrome one was used to display the dialogues between the

user and the computer, i.e. user commands and system messages. Berry [3] introduced a one-line diagram to display a portion of the whole power system so that various system quantities could be displayed to inform the user of the association between geographically related machine groups.

The current Bath power system simulator [1] has the following advanced features:

1. The power system under study is huge, i.e. the whole of the British National Grid Transmission System which is made up of a network of more than 400 busbars and 1200 lines. A large amount of simulation data is produced. If too much data is presented to the user, a mental bottleneck would be produced. If, however, too little data is given, the user is unable to analyze the power system. Besides, the physical limitation of how much data can be displayed clearly within a screen also imposes problems on the user interface.

2. The simulator is to be run in real time [6]. Special tools and design concepts are needed to help the user in carrying out an efficient power system study. This problem is quite unique among all the simulators as discussed above.

It is precisely under these two circumstances that the MMI is implemented.

6.3 System Configuration

The following gives a description of the hardware and software used for the implementation of the MMI.

[6] Refer to Glossary

6.3.1 Hardware

The power system simulator hardware [7] is made up of the followings:

- three transputer boards, each with one T800 transputer [9], 1 MByte of RAM
- one transputer link topology card with one T800 transputer, 1 MByte of RAM
- one graphics board, with one T800 transputer, one VSC graphics processor [10] and 4 MBytes of video RAM.
- one I/O card with a Philips 68070 [77] microprocessor
- one 45 MByte hard disc
- one 800 KByte floppy disc
- one console (a monochrome display and a keyboard)
- one bit-mapped colour monitor, of resolution 720 x 480 pixels
- one pointing device, in this case, a mouse
- one keyboard

6.3.2 Software

The MMI is implemented in a very portable way. The only machine dependent part is contained within a single program module. With a system supporting the

[7] Refer to Chapter 4 for a comprehensive description of the hardware

X Window System [7] and Helios Operating System [8], the MMI can be ported across easily. Fig. 6.1 illustrates the the software hierarchy of the MMI.

The organizations of the software can be broken down into the followings:

- Athena Widget Set, Xt Toolkit and X Lib are used to implement most of the MMI.
- The programming language used to implement the MMI is C [84].
- The MMI is made up of a number of program modules which can be separately compiled and then linked together. A full description is presented in Section 6.5 (The Structure of the MMI).
- All hardware dependent routines are contained within a single program module to make the MMI more portable.

6.4 Conceptual Model

The Real Time Power System Simulator is organized as in Fig. 6.2.

The simulator is split into four layers, namely the **User Interface, Application Interface, Simulation Code** and **Data**. The function of each layer is discussed below:

6.4.1 User Interface

A user does not interact with the simulation code and data directly, but rather all the I/O are done via the User Interface. In other words, the simulator is protected and insulated from human errors.

This layer is implemented using the X Window System and is responsible for:

1. presenting the "look and feel" of the simulator
2. translating all the external input from the user into a set of predefined calls to the Application Interface. The user's requests are treated as asynchronous events and an event handler residing in the User Interface is used to intercept and process them before re-directing them to the application interface.
3. processing the output data from the Application Interface to the user in a predefined format which is specified by the User Interface.

6.4.2 Application Interface

The Application Interface is a library of function calls. It is responsible for servicing all the requests from the User Interface by either reacting with the data directly (reading and saving data) or with the Simulation Code which is made up of a number of calculation tasks executing in parallel, e.g. reading the voltage level of a certain busbar from the network server.

6.4.3 Simulation Code and Data

The Simulator is organized into a parallel network server and a number of machine servers which are distributed around the Transputer Network. The machine servers are running independently in parallel and will communicate with the network server only when it is necessary. The Application Interface accesses the Simulator by sending requests to the network server. This is a one-way communication, i.e. the network server is passive while the Application Interface is activated by the User Interface which is itself event driven. The network server can not interfere with either the Application Interface nor the User Interface, except when the whole Simulator is failing and requires urgent user response. This situation rarely occurs and so the user is not disturbed too often.

6.4.4 Virtual Machine

The partitioning of the Simulator into 4 layers forms a virtual machine in the user's mind. The User Interface serves as the front-end of the machine, with the Application Interface as the I/O processor and the Simulation Codes running the engine. The advantages of this virtual machine are:

1. A user need not be aware of the internal organization of the simulator. The "look and feel" of the system is dependent only on the User Interface. This implies that the Simulator could be modified substantially in the Simulation Code and internal simulation data structures without the user ever noticing the difference. Hence, the cost of maintenance and upgrade is significantly reduced. As the user's own conceptual model of the Simulator is simplified this leads to an reduction in user anxiety.

As a user is saved from learning the underlying operating system and hardware, future upgrade of the simulator does not require another learning period from the user, who will already be familiar with the external interface. Such consistency helps the user in gaining confidence in running a new Simulator and reduces user errors when switching from one system to another.

2. Modifying the User Interface alone can give the Simulator a different "look and feel" if required. This can be done with minor changes to the external interface layer only, without ever touching the Simulation Codes.

   If a user is comfortable with a certain graphics environment, having spent a substantial amount of time with it, the transition period could be a lot smoother if the simulator has similar "look and feel" as the other system.

3. Because a user cannot interact with the Application Interface and Simulation Code directly, the system can anticipate the possible behaviours of the user. All the user's requests are handled via the User Interface which has only a limited amount of states and function calls. Hence, the simulator is less prone to human errors and is therefore more robust.

4. The User Interface is not tied up to the Data directly (other than file formats). Hence, restructuring the Simulation Data to suit future calculations algorithms does not require the User Interface to be modified. This ensures the consistency of the "look and feel" of the Simulator.

5. As the Simulator is split up into different layers, research can be diverted into implementing the Interface and Simulation Codes independently. Each researcher can upgrade a program without the fear of upsetting another, as long as a set of protocols is agreed beforehand, i.e. a predefined application interface is used.

Separating Interface from Application provides maximum flexibility and portability of the Simulator.

6.5 Structure of the MMI

The MMI is made up of a number of program modules:

1. A **Master Module** is responsible for:
   - initializing X Window
   - initializing internal MMI data structures
   - initializing the initial MMI screen
   - providing connections between menu buttons displayed and their associated functional modules.
2. **Function modules** which are responsible for providing services to various menu buttons.
3. **Library modules** which are responsible for providing sharable facilities to backup function modules.

As X Window is asynchronous, i.e. event driven, the master module will go into an infinite loop pending user inputs after all initializations are finished.

6.5.1 Screen Layout

The display screen is divided into 3 areas:

1. A banner at the top represents the top level options.

2. A large rectangle just below the banner represents the display area.

3. A long rectangle on the right hand side represents the sub-menu options.

Other than some pop-up menus (windows), no windows are overlapping. This gives the user a clear view of the system who is free to choose to look at different menus and their corresponding displays. It places less burden on the short term memory of the user.

6.5.2 Menus

There are basically two types of menus:

1. **Top Level Menus**

   They are always visible and selectable. The main functions they provide are:

   - **Network** - which displays a one-line network diagram of the power system under study.
   - **Time History** - which displays the time history plots of some of the power system variables.
   - **Operations** - which provides a menu to modify the simulator's behaviour.
   - **Set Points** - which provides a menu to change or examine various power system variables.

- **Faults** - which provides a menu to select a fault file under a specific directory.
- **Help** - which provides a hyper-text style help menu.
- **Quit Simulator** - quit the simulator.

Only a maximum of two options can be selected at a time in order to avoid confusion for the user. The "Network" and "Time History" options are mutually exclusive, i.e. only one of them can be selected at a time. The functionalities of these options are explained in details in Chapter 7.

2. **Sub-Menus**

   Most top level menus have a set of sub-menus associated with them. They provide further specific functions related to a top level menu. In this way, a user is protected from being presented with too many menu options at a given time, i.e. to ease the Interface Bottleneck. Besides, some top level menus are not compatible so that they cannot exist together. For example, page switching (discussed later in Section: Animation) for the network's animation cannot co-exist with the time history plots. Once a top level option is selected, its related sub-menu will be brought up. In some cases, these are in the form of a pop-up menu, visible only when requested by the user's direct-manipulations over a graphical object.

6.6 Interaction Style

The MMI adapts the WYSIWYG and Direct manipulation philosophies. A user does not have to memorize any commands. (However, if he wishes, he can type commands at the console.) The MMI is passive in nature. Nothing, other than the simulation, will happen unless the user has issued a command. All commands

available are displayed on a menu. A user just have to use a mouse pointer to click on a menu button in order to:

1. issue a command
2. change a power system parameter via a software calculator, appearing in the form of a pop-up menu
3. select an item, e.g. a file

In the one-line diagram, a power system entity, (e.g. a busbar), can be selected by clicking the mouse pointer over the graphics item representing it. Hence, the user is made to believe that he is directly manipulating a power system object. As graphics provides more vivid information about the entity and its surrounding environment, (e.g. the connection between busbars), it is a much better method than memorizing the abstract name of a busbar and typing commands at the console. By using a mouse pointer, a user can make use of most of the MMI, except in a few cases when the keyboard is needed to input a file name.

6.6.1 Feedback

In order to assure the user of his action, visual feedback is provided. For those actions which require a long execution time, the cursor of the mouse pointer is changed to a watch, indicating that the system is now busy processing the last command. At the same time, the user is prohibited from issuing any more commands. This is because the old and new commands might be conflicting. Once the last command has been processed, the cursor then changes its shape back to a large "X" and the user is allowed to issue new commands again.

6.7 The Use of Colour

Because the VSC system provides only 16 colours, which can be displayed simultaneously, a simple colour scheme is used throughout the design of the MMI. Colour is used in two main areas:

1. Menu
2. Data Display

6.7.1 Menu

The use of colour occurs in the following areas:

- Menus of different levels or functionalities have different background colours. This gives the user extra information about the nature of a menu.
- All menu options within the same menu have the same background colour so that a user can associate them as belonging to the same group.
- Black is used as the border colour surrounding each menu. This effectively highlights a menu and makes it stand above its surroundings.
- All menu labels are drawn in dark blue colour. By experiment, it was observed that this colour stands out on all the remaining 15 background colours. This also causes less confusion for the user as the background colour of a menu alone is enough to give him information about a menu.

6.7.2 Data Display

The use of colour lies in the following areas:

- Black is used as the background colour as the remaining 15 foreground colours can be distinguished easily. This provides more choices of colours for representing different data.
- Different categories of power system entities have different colours. For example, a busbar is drawn in dark blue while an electrical line is drawn in cyan. Busbars of different voltage levels are drawn in different colours.
- As animation is used, colour is not used to represent dynamically changing data, except where animation alone is inadequate, e.g. in a PQ-Chart, the moving pointers change colour depending on whether the associated machine is generating or absorbing power.

6.8 Animation

Animation can be used to display simulation results dynamically so that the power system's states can be monitored continuously. As well as giving the user information on what the system was doing a short time ago, animation also tells the direction that the system is presently going. It is in contrast to the traditional time history plots, which tell the past position of a system but not its future direction.

In order to achieve reasonable animation, necessary criteria must be defined. In an engineering application, animation must satisfy the following:

1. Flicker Free. If the animation is unpleasant to look at, it is very difficult to examine the data.

2. Fast Display. The frame rate, i.e. number of pictures displayed per second, must be reasonably fast so that a minimum of simulation data is lost. According to experience, a figure of about 10 is just enough to keep up with the data updating rate of the present power system simulator.

For the second criterion, it is enough to devise a good algorithm in updating animation data. The following algorithm illustrates the idea:

1. **Get Power System Data**
2. **Use this data to calculate the new information for animation**
3. **Erase the previous picture**
4. **Draw the new picture with the new data**

Steps 1-4 are repeated indefinitely.

However, the above algorithm is not enough to achieve flicker free animation. The intermediate stage, after the graphics have been removed and before they are redrawn, causes blanking of part of the screen. The rapid drawing and redrawing causes these blanking parts to blink in front of the user because the residue image of the last image interferes with the "empty graphics". No matter how fast the graphics are updated, the human eyes are irritated because of this flicker. There are 3 methods to solve the problem:

1. **Double Buffering**

2. **Colourmap Manipulation**

3. **Selective Updating of Graphics**

6.8.1 Double Buffering

There are two pages of graphics, i.e. two chunks of memory the size of each is enough to hold the whole bitmap of the screen. One is called the visual page and the other is called the active page. The visual page is displayed to the user, and the active page is updated in the background, i.e. hidden from the user. Then, the pages are switched over. This process continues indefinitely. This method is sometimes called **Page Flipping**.

Take the example shown in Fig.6.3.

In practice, the switching is done by switching a certain register within the graphics processor so that the visual and active pages are interchanged according to the value of that register. As the operation involved is extremely fast, there is no blinking on the graphics screen and the human eye is fooled into seeing a screen of graphics updated smoothly. Of course, if the graphics system does not support more than one page, page flipping is not possible.

In order to facilitate double buffering, some modifications to the previous algorithm are needed.

There are now two pages of graphics, each of which is updated independently and a separate link-list of graphics must be associated with each page. Otherwise the pages will be mixed up with each other as new system data is produced for each

frame.

The following is a modified version of the algorithm:

1. **Blank Visual Page**
2. **Get Power System Data**
3. **Draw Graphics on Active Page**
4. **Switch over Visual and Active Pages**
5. **Get Power System Data**
6. **Use Data to calculate the new information for Active Page**
7. **Erase Active Page Graphics**
8. **Draw Active Page with new information**
9. **Switch between Active Page so that Active Page becomes Visual Page and vice versa**

Steps 5 - 10 are repeated indefinitely.

6.8.2 Colourmap Manipulation

The appearance of a colour on a screen is determined by its corresponding pixel value. A pixel value is an index used to identify a particular colourcell within a colourmap. For instance, the colour "red" looks red on the screen because its associated pixel value is 224, i.e. 111 000 00 for a 8-bit per pixel colour display.

It means that the graphics processor is using 8 bits to describe a pixel value. The most significant 3 bits indicate the intensity of one primary colour, red; the next 3 bits - green; and the least significant 2 bits - blue. A pixel value must not be confused with its corresponding colourcell, e.g. a pixel value of 1 will select the first colourcell. Therefore, by setting different bits of a pixel value, the appearance of the colour associated with that pixel value will be changed accordingly. This system now has $2^8 = 255$ colourcells here, which can be displayed simultaneously on the screen. However, the maximum number of colour that can be produced for a display, which has 8 bits available for each primary colour, is 256^3 (about 16 million).

Assume that there are 255 pixel values and with two pages of graphics. Here, a page is only a software entity, i.e. a link-list is used to describe each page. It is possible to divide the colourcells into 2 sets, with 127 colourcells for each page, page 0 and page 1. Set 0 comprises the colourcells used by page 0 and set 1 contains those used by page 1. The remaining 1 colourcell is used for the background colour in both pages. The following algorithm illustrates the principle:

1. **Get Power System Data**
2. **Use these data to calculate the new information for Page 1**
3. **Set all the pixel values in Set 0 to background colour so that Page 0 is now erased.**
4. **Update the link-list of Page 1 with the new data**
5. **Draw Page 1 with Set 1 colourcells so that Page 1 appears**
6. **Switch over Page 0 and Page 1, Set 0 and Set 1**

If, however, the maximum number of colour that can be displayed is limited, there will not be sufficient colours for each page to convey information.

6.8.3 Selective Graphics Updating

Flickering occurs most severely when the power system model is in a steady state, i.e. there is not much change in the animation graphics and the computer system just erases a piece of graphics and redraws it later. If the original algorithm is modified in such a way that only the graphics which have been updated are changed, then computer time is saved and also the graphics are not erased as often as before. In a steady state, the picture is in fact intact. However, during transient state, nearly all the graphics are changed. Then, the above checking of graphics will slow animation down and actually make the flickering worse.

The problem can be reduced by redrawing a piece of graphics immediately after it has been erased. In this way, none of the graphics is left "blank" for too long.

6.8.4 What the MMI has adopted

The MMI adapted the method of double buffering for animation for the following reasons:

1. The VSC system supports two page operation in hardware.
2. The software algorithm is the simplest in all three methods.
3. Colourmap manipulation is troublesome and wasteful in pixel usage.

4. In a power system transient state, the complicated method used in updating graphics selectively does not work well.

Unfortunately, X does not yet support double buffering. Therefore, it is necessary to derive a method to combine the capability of the VSC with X.

6.8.5 Double Buffering and X

For the animation program, a rectangle of the screen is controlled by the VSC directly. X is only aware of its existence but has no access to the graphics there. A set of low level graphics, i.e. writing directly to the screen memory, is used to draw animation graphics. X is left to control the other aspects of the MMI, e.g. reporting the position and state of the mouse pointer over the rectangle so that the user can select any graphics inside it. It looks as though X knows where the graphics are, even though they are not drawn by it.

The following algorithm is used:

1. **Use X to initialize screen**
2. **Define a rectangle on the screen for VSC**
3. **Draw all those graphics, which will not be changed during animation, on the Visual Page**
4. **Copy everything on the screen, i.e. Visual Page, to Active Page**
5. **Use Double Buffering as suggested in the above section**

As the VSC does not support partial page switching, i.e. part of the visual page is switched over with part of the active page, the action of switching pages is restricted only when the mouse pointer is within the rectangle. When the mouse pointer is outside, the screen behaves as if only one page exists so that only the last updated visual page image and X graphics are visible. Otherwise, the graphics produced by X will be only visible on the visual page but not on the active page. Those graphics will appear to be blinking as the two pages are switched over repeatedly.

The advantage of the above method has the best of both world, i.e. smooth animation (double buffering) and X for MMI. However, this combination does have some drawbacks:

1. Inconsistent application style. X does not know VSC and VSC does not know X. The MMI has to make different allowance for the two systems.

2. X is asynchronous. Graphics drawn by X do not appear instantly. If these graphics are not copied to the active page, they will appear to be blinking when page switching takes place later. It is very difficult to control the timing and it is usually done by trial and error.

However, smooth animation is too important and the above short-comings can be tolerated by careful planning and programming. Experience shows that a user quickly adapts to the behaviour of this MMI and is not deterred from using the simulator. Hence, a good engineering compromise has been reached.

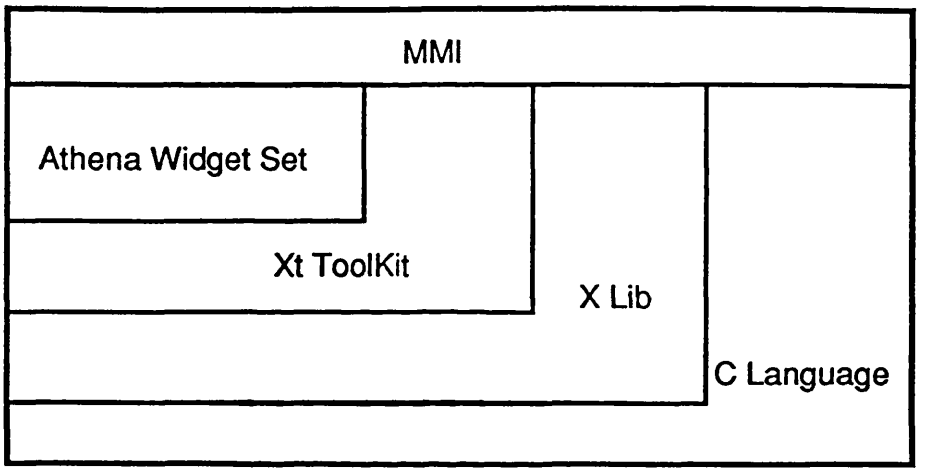


Figure 6.1: Software hierarchy of the MMI

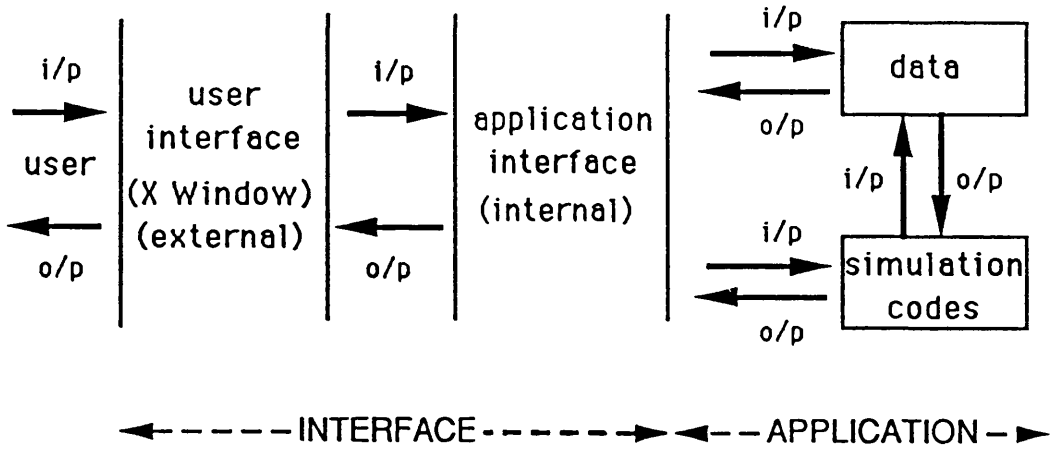


Figure 6.2: Organization of the real time power system simulator

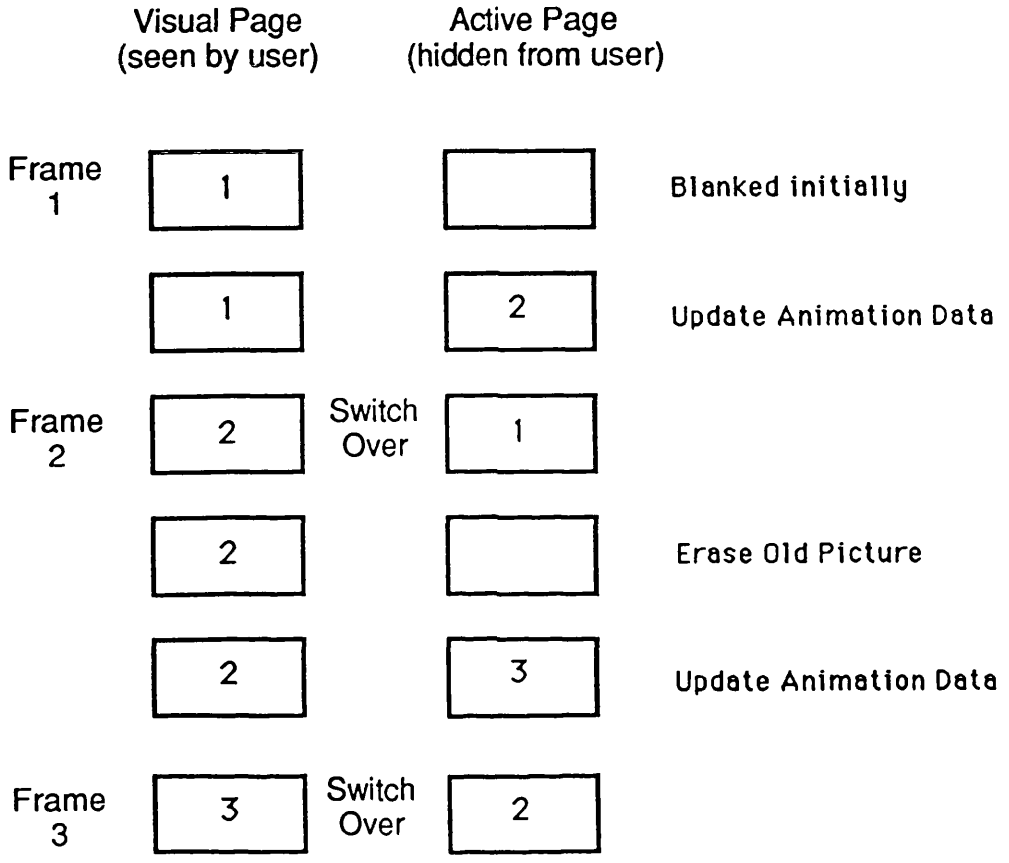


Figure 6.3: An example showing the effects of double buffering

Chapter 7

Features of The Interface

7.1 Introduction

The design theories, software hierarchy and various interface implementation techniques have been described in the last chapter. It is therefore appropriate to now present the various features of the MMI so that its basic functionalities and characteristics can be understood.

After the procedure of starting up of the MMI is described in the next section, the reader is guided around the system and the features of the MMI are unveiled in the subsequent sections. An example [1] is used as a vehicle to help to drive home the ideas. Descriptions of X Window [2] [64], Helios operating system [3] [8] and hardware [4] have been presented in previous chapters. A full description of the Simulator is given by Berry [3].

[1] For simplicity's sake, a four machine system [2] serves as the power system to be studied. Appendix 5 contains the relevant features of the four machine simulation used to demonstrate the implementation of the MMI.

[2] Refer to Chapter 3

[3] Refer to Chapter 4

[4] Refer to Chapter 5

7.2 Getting Started

The user must create the following directories under the same parent directory in which the object codes of the Simulator and the Interface reside:

1. **study** - which contains the power system study files used by the Simulator.
2. **help** - which contains the help files used by the Simulator.
3. **plot** - which contains the time history plot files saved in some earlier runs of the Simulator and which are used by the Time History option.
4. **fault** - which contains the files defining various power system sequences, e.g. a busbar fault followed by the opening of a circuit line. The files are used by the Fault option.
5. **pic** - which contains the files defining the pictorial representations of power system networks. The files are used by the Network option.

To run the four machine study, the user has to type the following text on the console:

% source m4start

where % is the Helios shell prompt and m4start is the shell script file containing all the necessary commands for bringing up the Simulator and the Interface. A description of the file m4start is given in Appendix 1.

Fig. 7.1 shows the screen after initialization is completed.

The system is now ready for user input and either one of the Top Level Menus can be selected.

7.3 Network

The Network option is responsible for presenting a one-line diagram of the power system to be studied and provides means for the user to use direct-manipulation, via a mouse, to interact with the system, e.g. issuing commands and getting information. There are two pop-up menus for that purpose, a Parameters Menu and a Meters Menu. A number of menu options, the Network Sub-menu options, are also available for providing further services.

Fig. 7.2 shows the screen of the Network option. Fig. H.5 shows a similar screen in colour.

7.3.1 The mouse

There are three buttons on the mouse. Basically, the mouse is used to select an option from the Network Sub-menu and to interact directly with the graphical objects on the one-line diagram.

For the first function, only the left hand button can be used to select an option from the menu.

For the second function, the three buttons have different meanings. Numbering from left to right:

- Button 1 is used to pop up the Parameters Menu and let the user change the parameters of the selected object.
- Button 2 is used to pop up the Meters Menu and let the user select a meter to monitor the behaviour of the selected object.
- Button 3 is used to tag an object. The colour of the label of the object will toggle between red and white. All objects of the same power system type and with red labels are presented together when the Vector Diagram is selected.

7.3.2 Parameter Menu

The user can change the parameters of three types of power system objects, i.e. machines, busbars or electrical lines. As the menu is popped up, the one-line diagram will stop its activity and animation is frozen. The Parameters Menu will change its menu of parameters according to the object selected. When a parameter is selected, a software calculator, in the form of a pop-up window, will be popped up. A menu bar is used to display the information of the selected parameter. Some basic calculator functions are provided to help the user in entering data. The user can enter a numerical value for that parameter either by using the mouse to click on the calculator or by typing keys on a keyboard. Once the user has finished with the menu, the one-line diagram will resume its activities and animation starts again.

Fig. 7.3 shows the screen of the Parameters Menu for a machine. Fig. H.9 shows a similar screen in colour.

Fig. 7.4 shows the screen of the Parameters Menu for a busbar.

Fig. 7.5 shows the screen of the Parameters Menu for an electrical line.

Fig. 7.6 shows the screen of a software calculator updating the avr value of a machine.

7.3.3 Meters Menu

The Meters Menu is used to present a menu of meters which are used to monitor the behaviours of a machine or a busbar.

Fig. 7.7 shows the screen of the Meters Menu.

There are four meters available:

- **Machine Diagram** - this presents a detailed description of the machine. A number of small meters are used to display the numerical values of the machine parameters.

  Fig. 7.8 shows the screen of the Machine Diagram. Fig. H.12 shows a similar screen in colour.

- **Vector Diagram** - this displays the vectors of the rotor angles of a group of machines or the vectors of the voltages of a group of busbars. A different colour is used to represent each vector.

  Fig. 7.9 shows the screen of the Vector Diagram displaying the vectors of the rotor angles of a group of machines. Fig. H.1 shows a similar screen in colour.

- **Time History Diagram** - this displays the time history plots of the states

of a machine. A panel of buttons is available for the user to examine the plot of any one of the ten states of the machine at a time. This is a simplified time history diagram. A more powerful graph plotter, the "Time History" option, is available as a Top Level option.

Fig. 7.10 shows the screen of the Time History Diagram displaying a plot of the rotor angle of a machine. Fig. H.8 shows a similar screen in colour.

- **Running Graph Diagram** - this monitors two parameters of a machine simultaneously. Up to eight seconds of the parameters' histories are displayed. As new data is produced, the currently oldest data is lost, operating on a First In First Out basis. Thus this diagram behaves like an oscilloscope. To assist the user in viewing the traces, facilities are available to freeze animation, expand or shrink the y-axis. The parameters available for selection are identical to those in the Time History Diagram. A button labelled "Run Sequence" can be used to start a pre-defined power system sequence. This is useful for examining the effects of a certain power system sequence on the power system network.

  Fig. 7.11 shows the screen of the Running Graph Diagram. Fig. H.11 shows a similar screen in colour.

7.3.4 Network Sub-menu

There are three types of option available:

1. **Global View Diagram:** At start up, only a portion of the power system network is shown in the large Network window and it is called the current Network Diagram. The Global View Diagram displays the whole power

system network. For a complex power system, the one-line diagram could be very large. It is thus inappropriate to have the whole picture displayed at the same time. The Interface will only display a portion of the lower left hand corner of the picture initially. Later, the user can pick up other part of the network by moving a rectangular frame around the Global View Diagram. Everything lies within the frame will be selected and displayed in the new Network Diagram. The current Network Diagram is shown in red while the rest of the system is drawn in other colours. A full description of how a one-line diagram is produced is given in Appendix 2.

Fig. 7.12 shows the screen of the Global View Diagram.

2. **Network:** When the Machine Diagram, Vector Diagram, or Running Graph Diagram is brought up, the Network Diagram is hidden from the user. The Network Diagram can be made active again if the Network option is selected.

3. **Sequence Specification:** It is possible to specify a sequence of actions to be applied to the power system. A detailed description of constructing a sequence is discussed in Appendix 3. The user can either specify the sequence interactively by direct-manipulation with the graphical objects on the one-line diagram or reading in a previously prepared text file. There are five options available for specifying a sequence:

   (a) *New Sequence* - this starts a new sequence. When this option is selected, the mouse pointer is changed to the shape of a hand. The user can change the parameters of a machine, a busbar or an electrical line one by one. Hence, only button 1 of the mouse is active.

   Fig. 7.13 shows the screen when a new sequence is to be selected.

   (b) *End Sequence* - this ends a new sequence and the mouse pointer is changed back to a large "X".

(c) *Table Sequence* - this displays a table of a previously defined sequence.

Fig. 7.14 shows the screen with a table of a given sequence.

(d) *Load Sequence* - this pops up a menu of text files. The user can load up a text file and use the content of it to specify a new sequence.

Fig. 7.15 shows the screen with a menu of sequence files.

(e) *Save Sequence* - this allows the user to save the current specified sequence into a text file for latter use.

Fig. 7.16 shows the screen of the Interface prompting the user for an output file name.

Fig. 7.17 shows the screen of the Interface prompting the user to input some optional data in specifying a sequence.

Fig. 7.18 shows the screen of the Interface accepting the input of duration time in specifying a sequence.

7.4 Time History

The Time History option is responsible for displaying the time history plots for the machines in the power system. There are two methods of displaying the plots:

1. Up to four graphs can be superimposed together on a large window, Graph Window, with a unique colour representing each graph.

   Fig. 7.19 shows the screen of four graphs displayed together. Fig. H.4 shows a similar screen in colour.

2. The large Graph Window is divided into four smaller windows, each displaying a graph.

Fig. 7.20 shows the screen of four graphs displayed separately. Fig. H.10 shows a similar screen in colour.

The advantage of using the two display methods is that the user can either compare the graphs together or study them in isolation.

There are a number of options available to assist the user in studying the power system:

1. **Colour** - The user can choose the colour of each graph separately.
2. **Grid** - The user can toggle this button to turn on or off a matrix of lines superimposed on the graphs.
3. **Save** - The user can save a graph into a binary file for later use.

   Fig. 7.21 shows the screen of the interface prompting the user for an output file name.
4. **Plot** - The user can plot a new graph on its own, and erase the last one.
5. **Same** - The user can superimpose a new graph on the existing graphs.
6. **Read** - The user can read in a binary file, saved in a previous session, and use it to display a new graph.

   Fig. 7.22 shows the screen of the interface prompting the user for an input file name.
7. **Table** - The user can pop up a window to display the numerical data of the graphs in a table for examination.

   Fig. 7.23 shows the screen of a table displaying the numerical data of a graph.

8. **Data** - The user can choose to plot the parameters of any one of the available power system machine.

   Fig. 7.24 shows the screen of the Interface prompting the user to select a machine.

9. **X-axis** - The user can change the x-axis of the graph to any one of the parameters available for plotting. The default is time, in second.

10. **Y-axis** - The user can pick any one of the parameters to use as the y-axis variable.

11. **variable** - The user can choose to plot the system, busbar or machine parameters.

7.5 Operation

Fig. 7.25 shows the screen of the Operation option. Fig. H.3 shows a similar screen in colour.

The Operation option provides the following facilities:

1. displaying the current log buffer size [5] in seconds.

2. displaying the current simulation time step [6] in seconds.

3. setting the reference angle of the rotor angle of a power system machine. The rotor angle can be calculated with reference to:

[5] A log buffer is a software buffer used to contain the simulation results of a power system machine. It is updated indefinitely when the Simulator is functioning.

[6] It is the time step used in calculating the numerical solution in solving the power system matrices.

- Mean rotor position - which is the average of all the rotor positions of all the machines in the power system.
- 50 Hz - which is the rotor angle of the slack busbar used in the power system network reference frame operating at a frequency of 50 Hz.
- Busbar voltage angle - which is the voltage angle of the busbar connected to the machine.

4. restoring the Simulator model to its previously remembered state.
5. remembering the current state of the Simulator model.
6. holding the Simulation Model temporarily.
7. releasing the Simulation Model from its previously frozen state.
8. changing the log buffer size via a pop up software calculator.

   Fig. 7.26 shows the screen of how the log buffer size is changed.
9. changing the simulation time step via a pop up software calculator.

7.6 Set points

The user can change the set points or parameters of some power system objects by two methods. The first is to use direct-manipulation to interact with the graphical objects on the one-line diagram from the Network option. The second is to invoke the Set Point option. In addition, the parameters of a governor and an avr can also be changed via this option. As direct-manipulation is then impossible, the user has to select whichever power system object by picking an entry from a table of objects displayed on a pop-up menu. Once an entry is

selected, the same input procedure as used in the Network option is used to update a parameter.

As well as changing the set points or parameters of some power system objects, a spread sheet is available to assist the user in monitoring some of these numerical values. The numerical data of the set points or parameters of a machine, busbar, electrical line, set, governor or an avr can be arranged into a matrix of data, in the form of a number of spread sheets. Only one spread sheet may be viewed at a time. Facilities are provided to:

1. sort the data in ascending order.
2. sort the data in descending order.
3. display the minimum values of parameters since starting up of the Simulator.
4. display the maximum values of parameters since starting up of the Simulator.
5. reset the minimum and maximum values of parameters.
6. update the spread sheet data in order to display new simulation results.

Fig. 7.27 shows the screen of the first pop up menu of the Set Points option.

Fig. 7.28 shows the screen of selecting a machine from a pop-up menu.

Fig. 7.29 shows the screen of selecting a busbar from a pop-up menu.

Fig. 7.30 shows the screen of selecting an electrical line from a pop-up menu.

Fig. 7.31 shows the screen of selecting a governor.

Fig. 7.32 shows the screen of changing a parameter of a governor. Fig. H.6 shows a similar screen in colour.

Fig. 7.33 shows the screen of selecting an avr.

Fig. 7.34 shows the screen of changing a parameter of an avr.

Fig. 7.35 shows the screen of a spread sheet of machines.

Fig. 7.36 shows the screen of a spread sheet of busbars.

Fig. 7.37 shows the screen of a spread sheet of electrical lines. Fig. H.7 shows a similar screen in colour.

Fig. 7.38 shows the screen of a spread sheet of sets.

Fig. 7.39 shows the screen of a spread sheet of avrs.

Fig. 7.40 shows the screen of a spread sheet of governors.

7.7 Faults

The user can either specify a sequence from the Network option or the Faults option. This provides a flexible means to enable the same function to be achieved

via different paths [7]. A table containing all of the available text files is popped up for selection. When a file has been selected, its content is displayed so that the user is reminded of the function of the file.

Fig. 7.41 shows the screen of the popped up table of files.

Fig. 7.42 shows the screen of the content of a selected file. Fig. H.2 shows a similar screen in colour.

7.8 Help

A hypertext [8] style help menu is used to guide the user in using the Interface.

Fig. 7.43 shows the screen of the initial help menu. Fig. H.13 shows a similar screen in colour. The title of the current help page is shown on the top right hand corner while the title of the last help menu page is shown at the bottom right hand corner of the help menu. The user can refer to the last menu page relative to the present one by clicking the button labelled "Last Topic".

The highlighted keywords can be selected by the user and they lead to further menu pages, giving more information about them, with additional keywords. The menu is divided into a set of interconnected menu pages with the highlighted keywords serving as some sort of gateways connecting the pages. By placing the label of an additional menu within the context of a help file gives the user extra information about the keywords and its relationship with the current help information [104].

[7] The importance of this idea is discussed in Chapter 2.
[8] Refer to Glossary

The method of creating a system of hypertext menu is given in Appendix 4.

7.9 Quit Simulator

The user can quit the Simulator by selecting this option. The large window displaying all the components of the Interface is popped down and the screen is restored to the default X Window background.

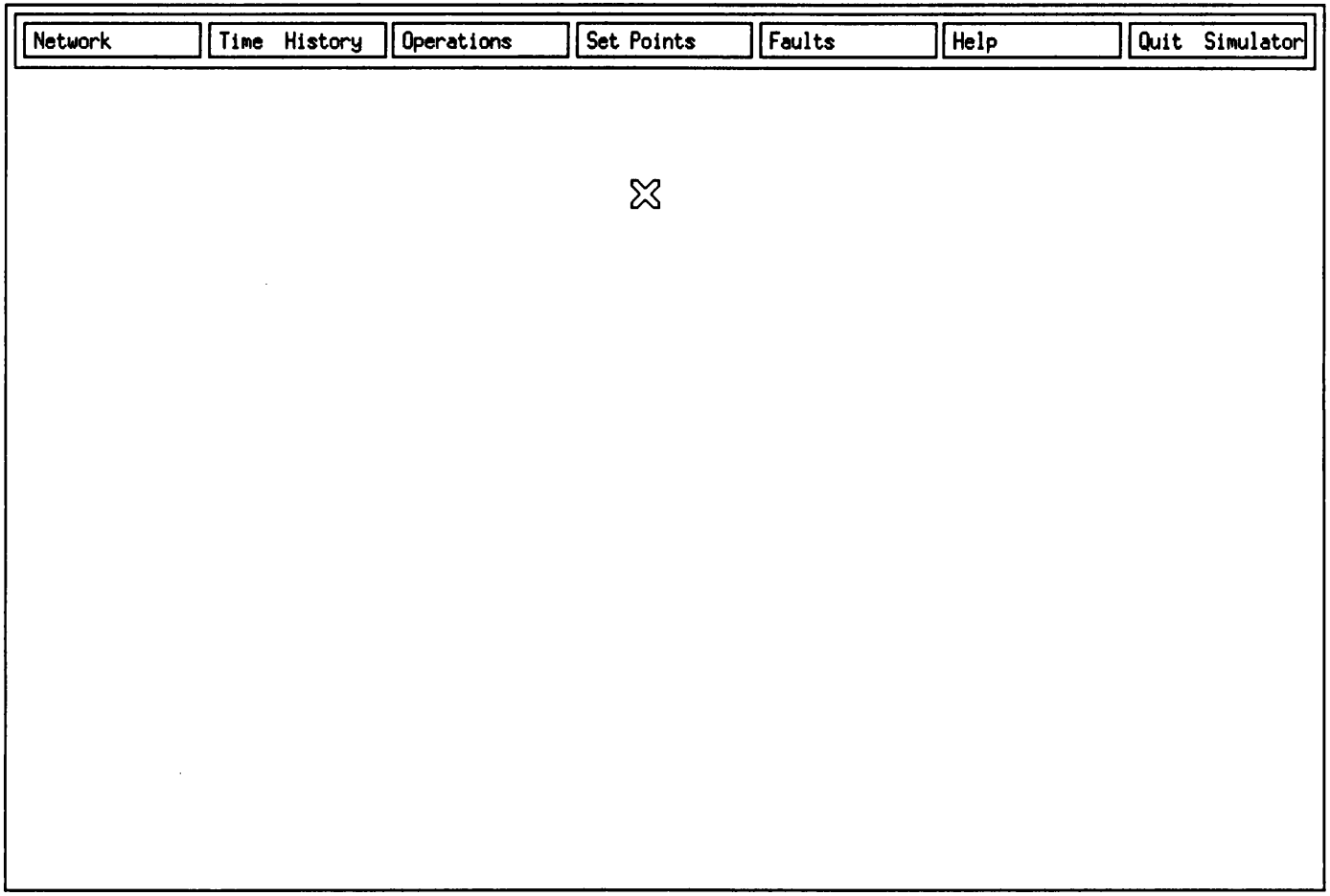


Figure 7.1: Start up screen of the MMI

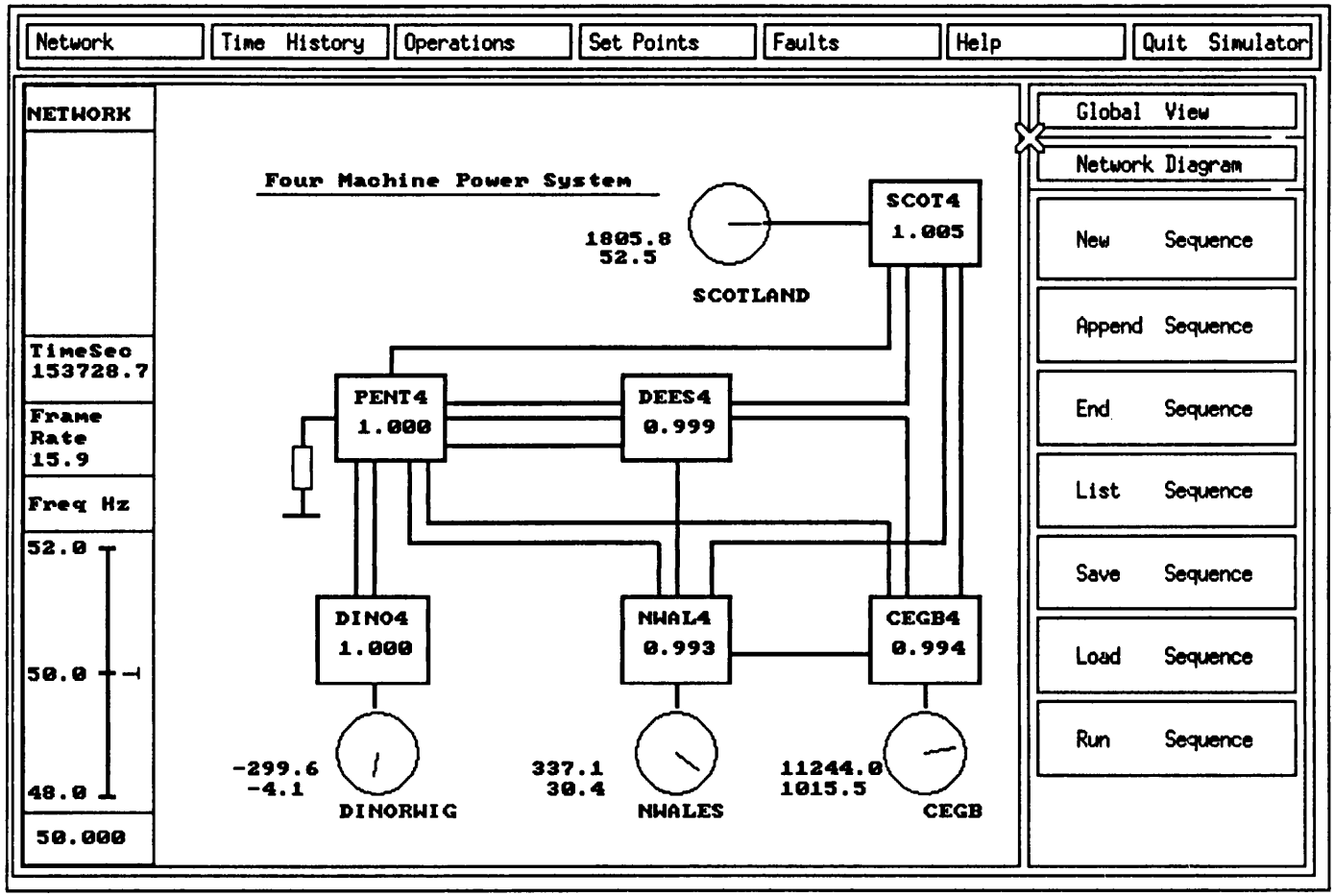


Figure 7.2: One-line diagram of the four machine study

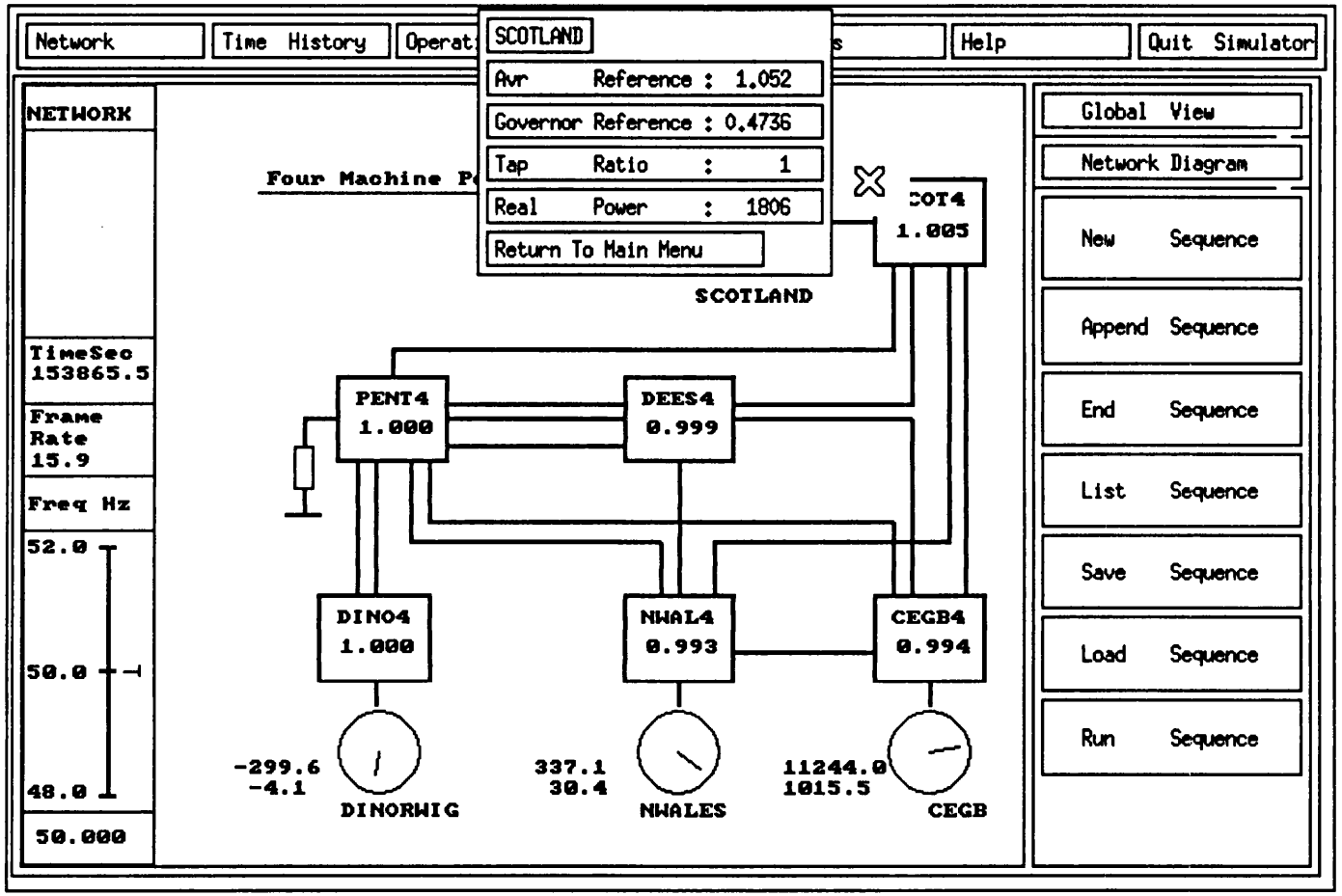


Figure 7.3: Menu of the set points of a machine

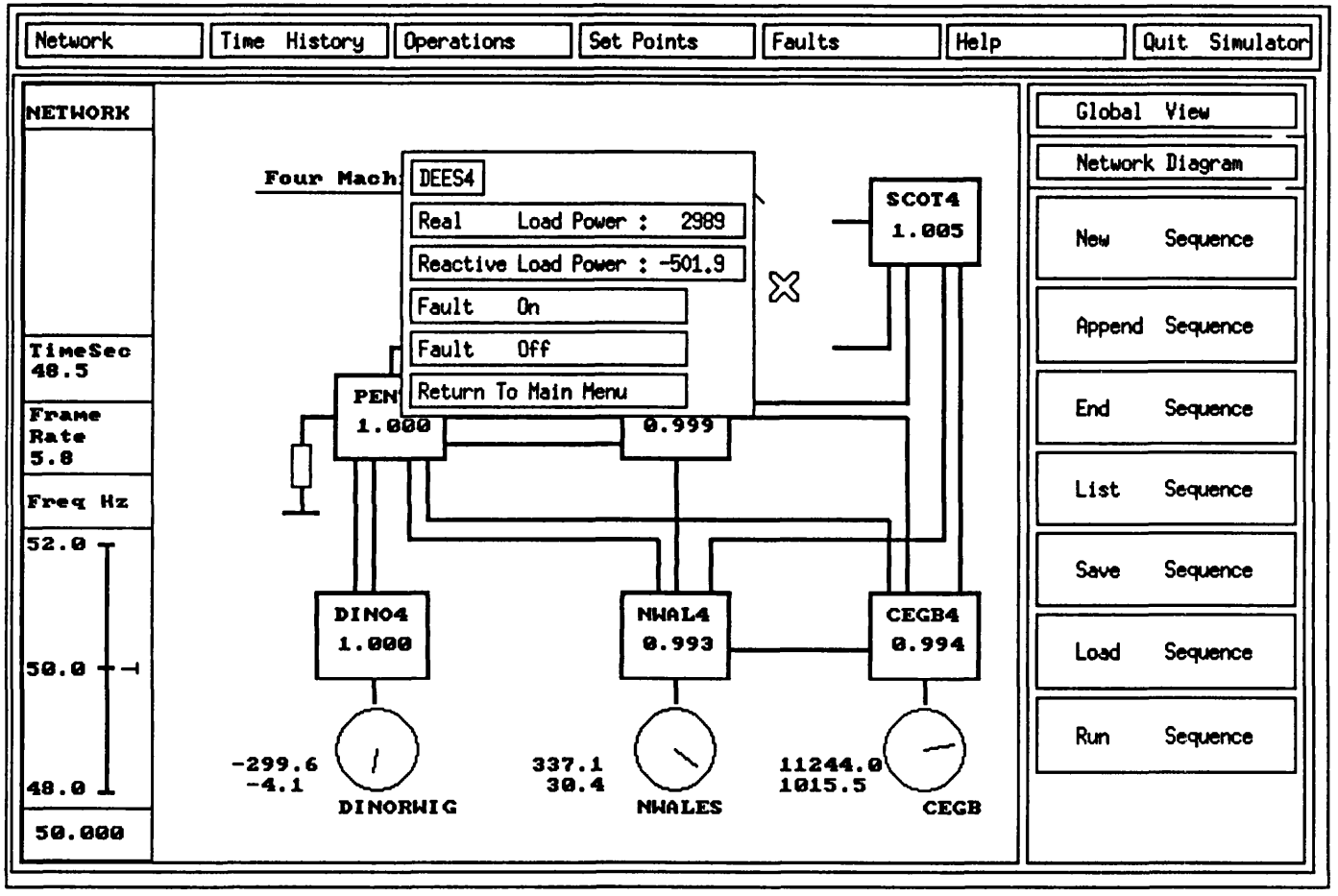


Figure 7.4: Menu of the set points of a busbar

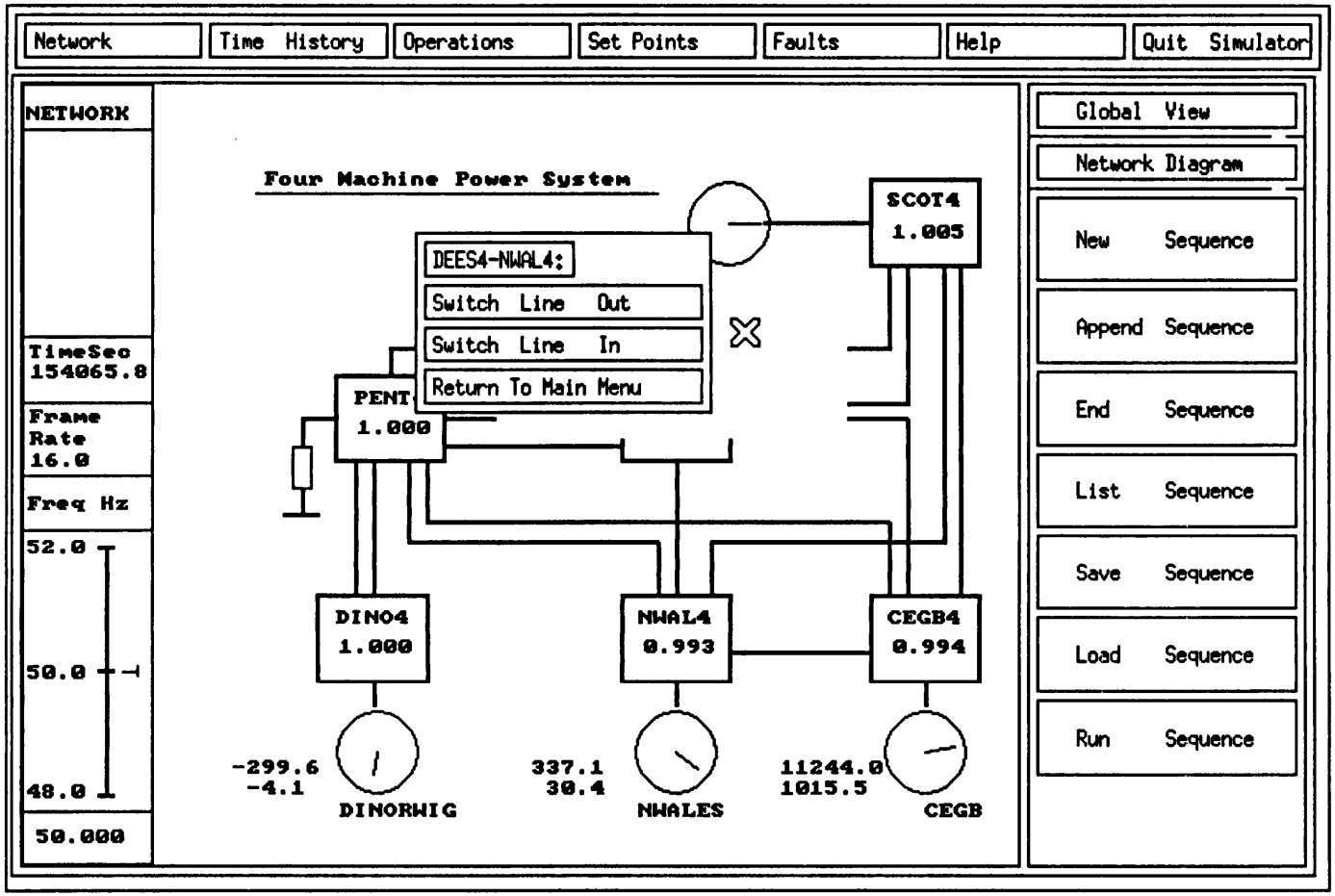


Figure 7.5: Menu of switching in/out of a line

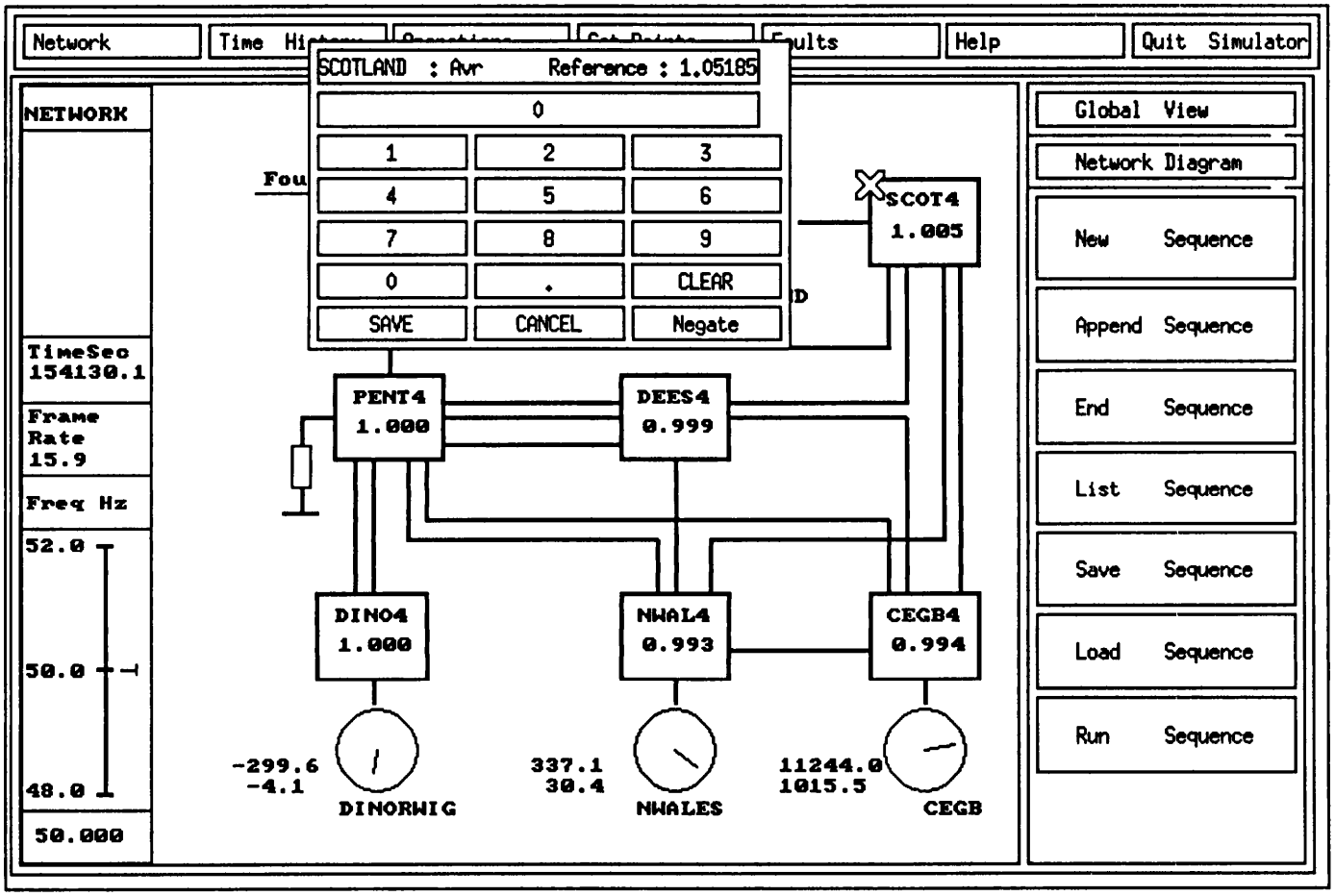


Figure 7.6: Modifying the avr reference of a machine

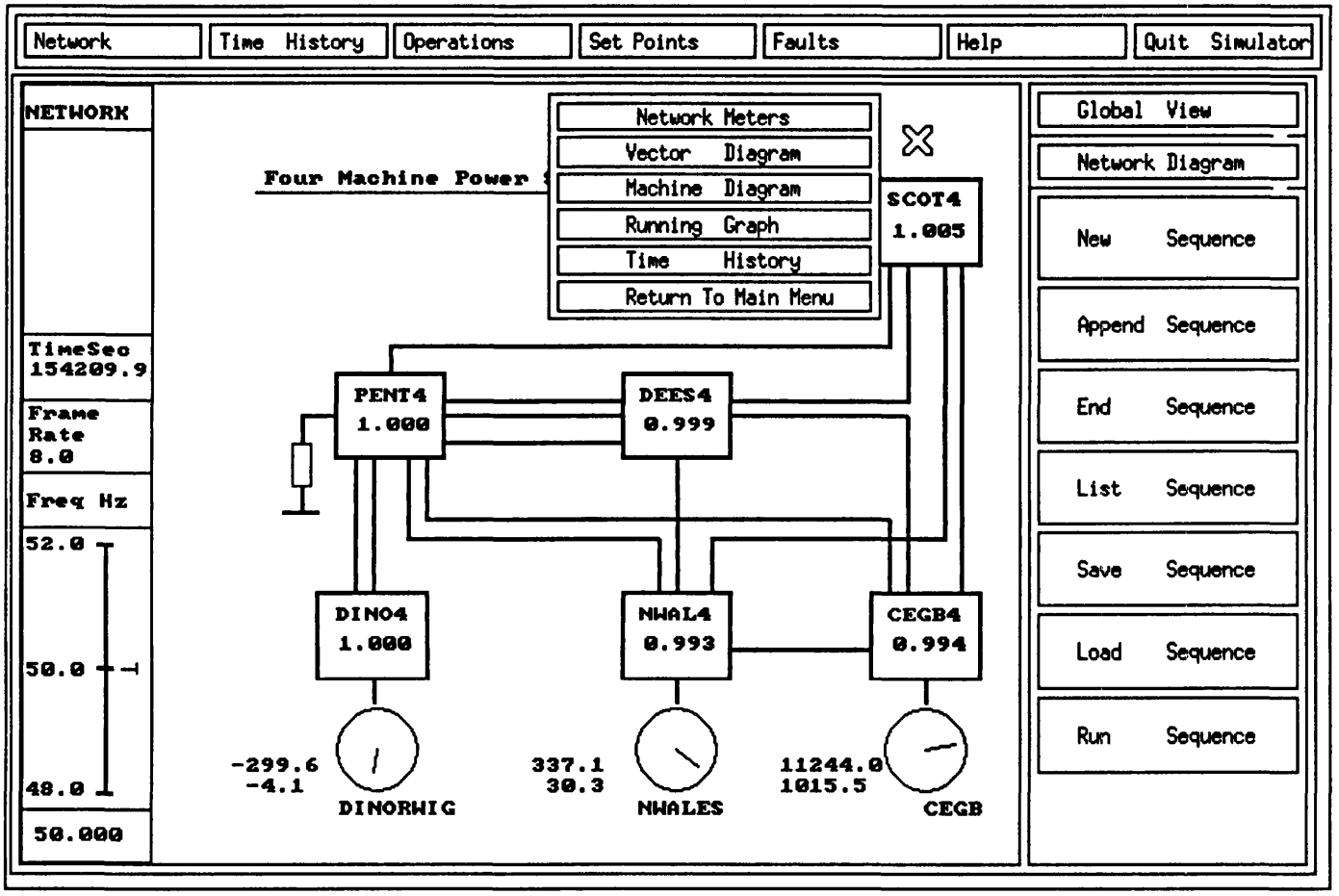


Figure 7.7: Menu of network meters

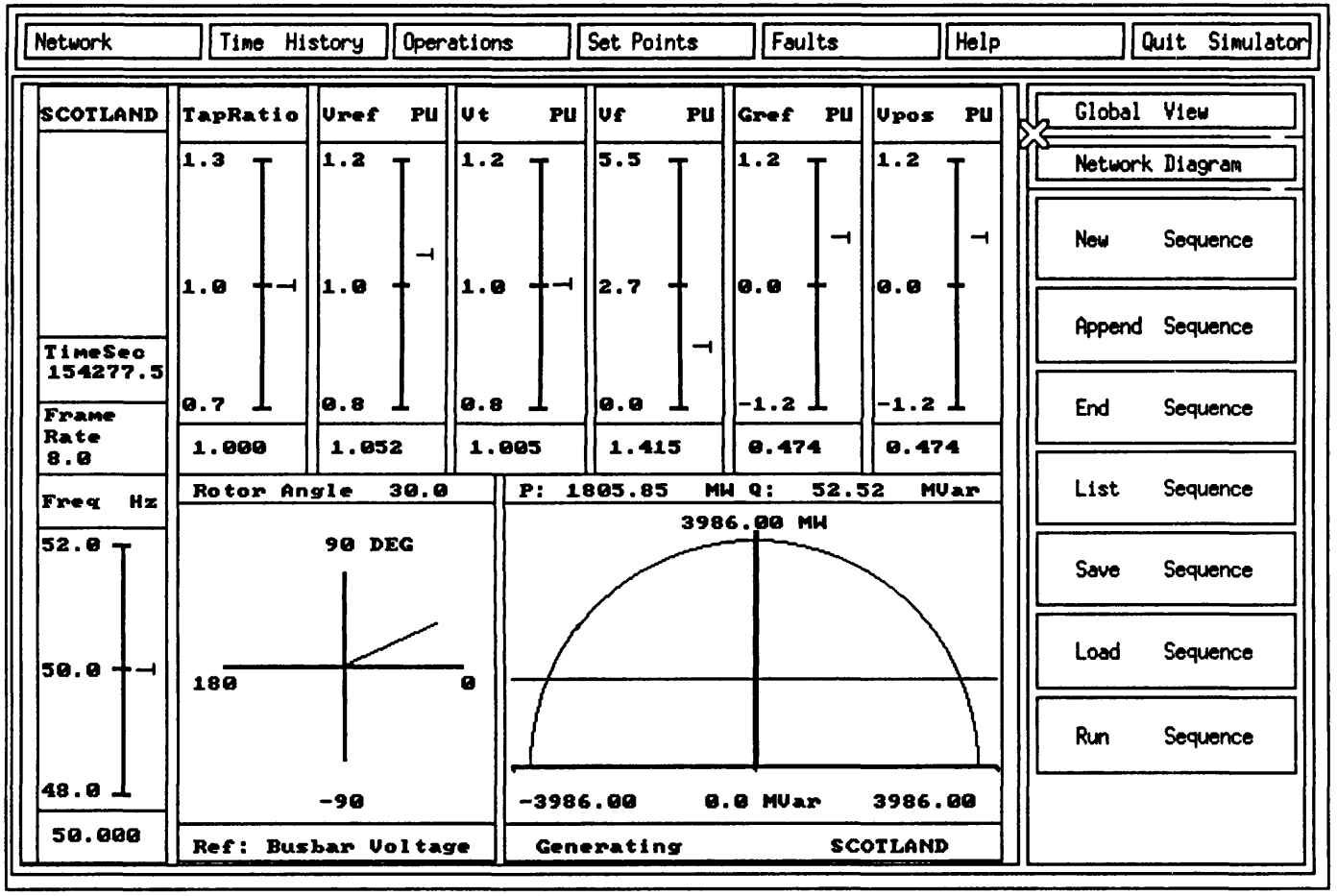


Figure 7.8: A detailed machine diagram

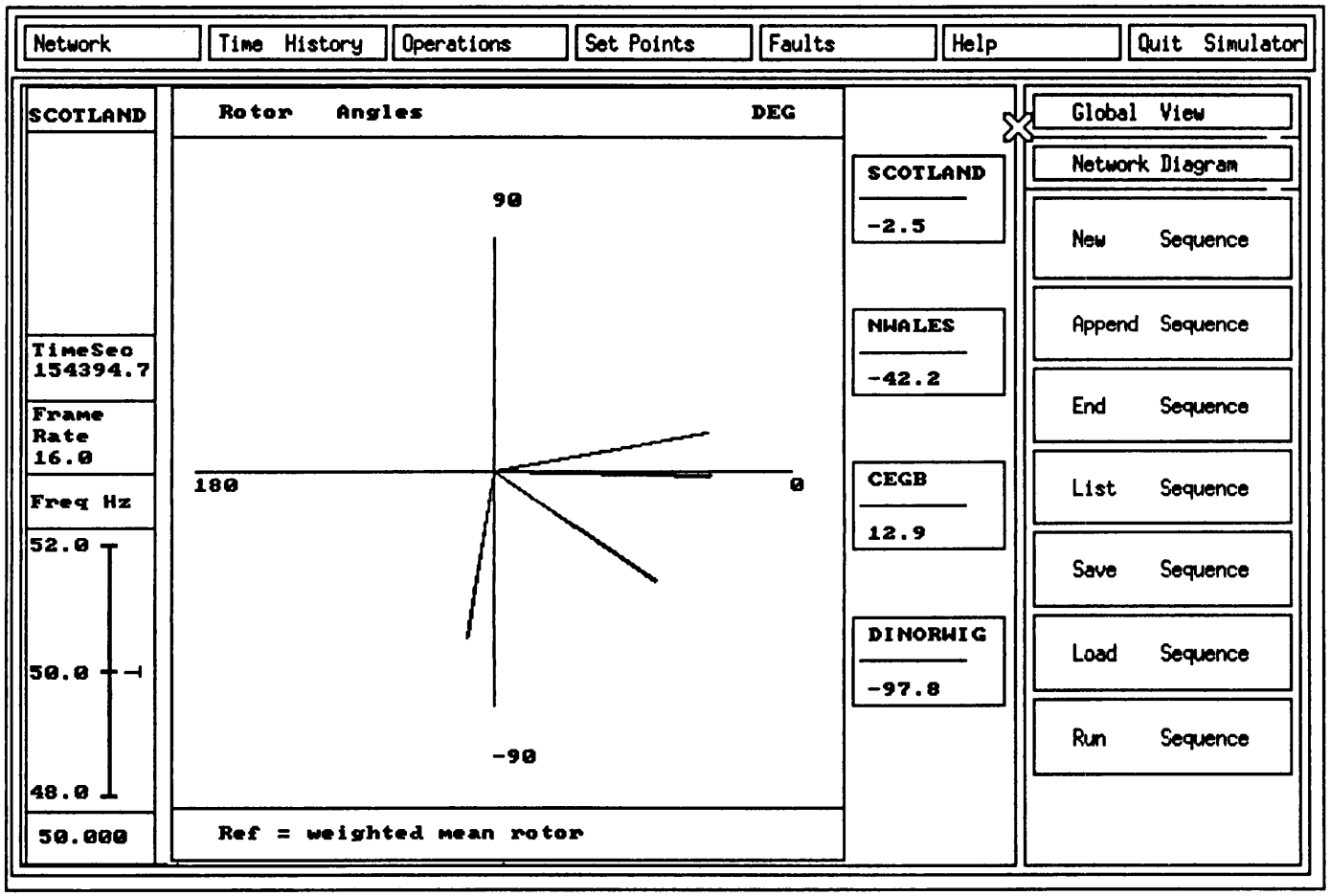


Figure 7.9: A detailed vector diagram of four machines

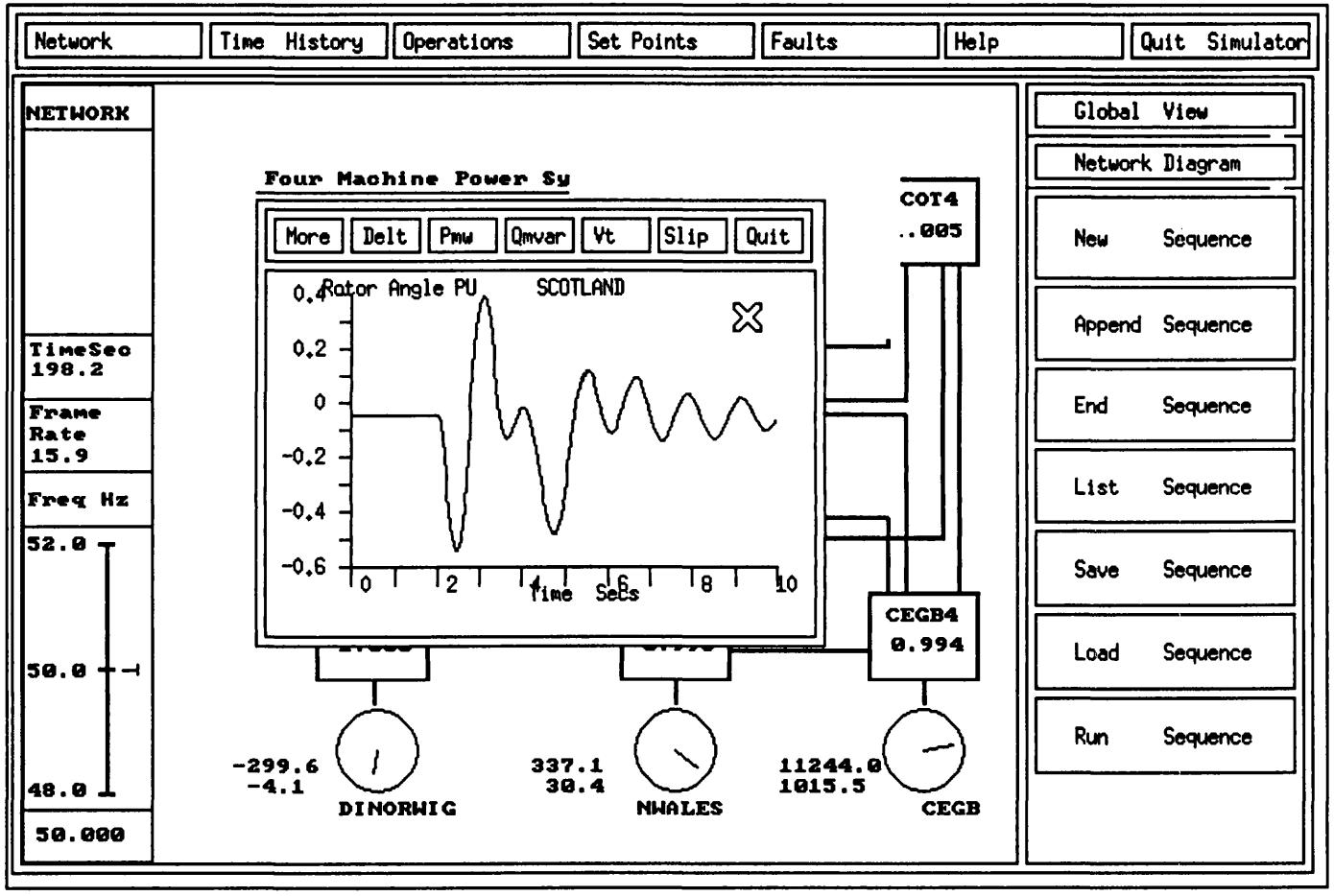


Figure 7.10: Time history plot of a machine

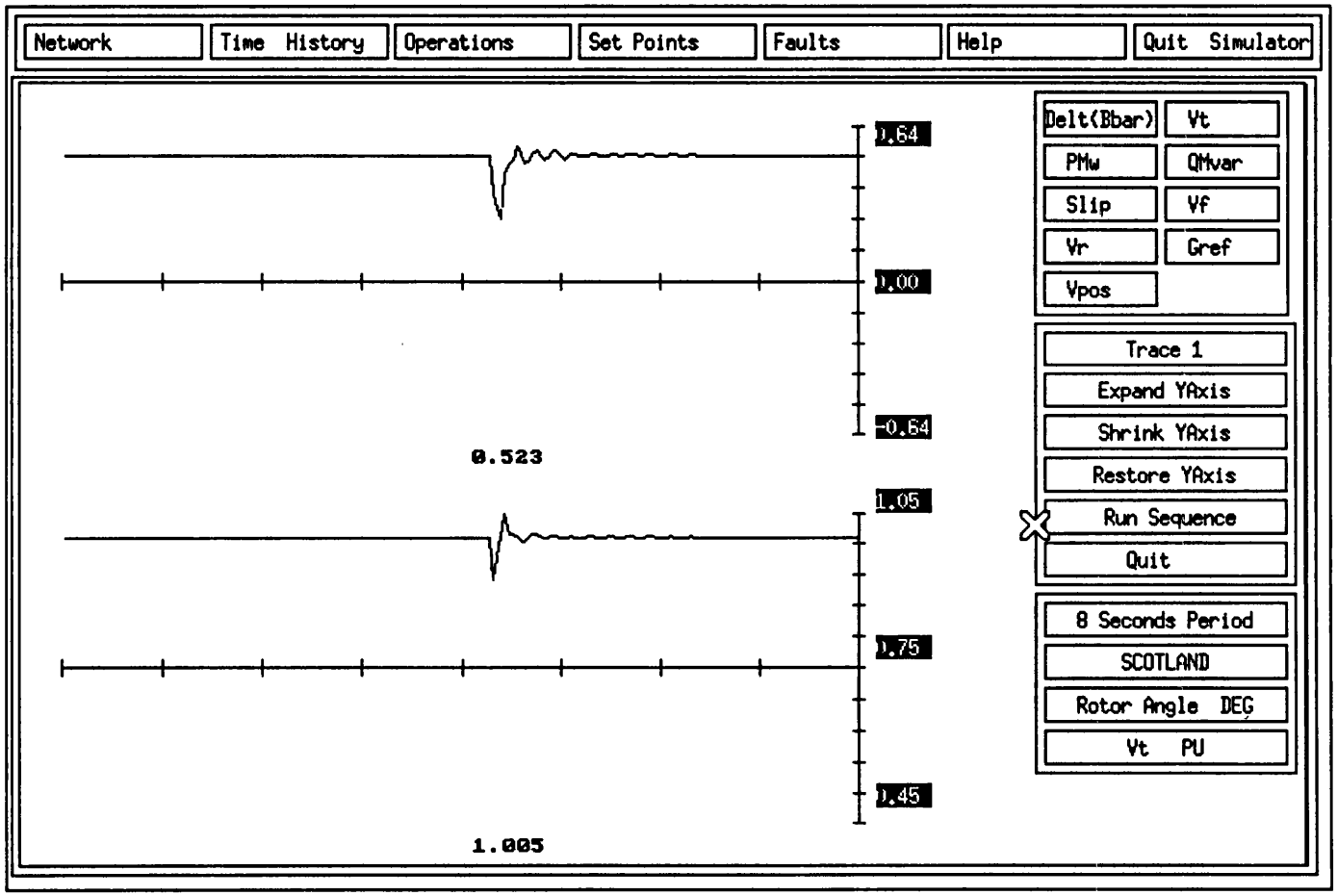


Figure 7.11: Running graph of a machine

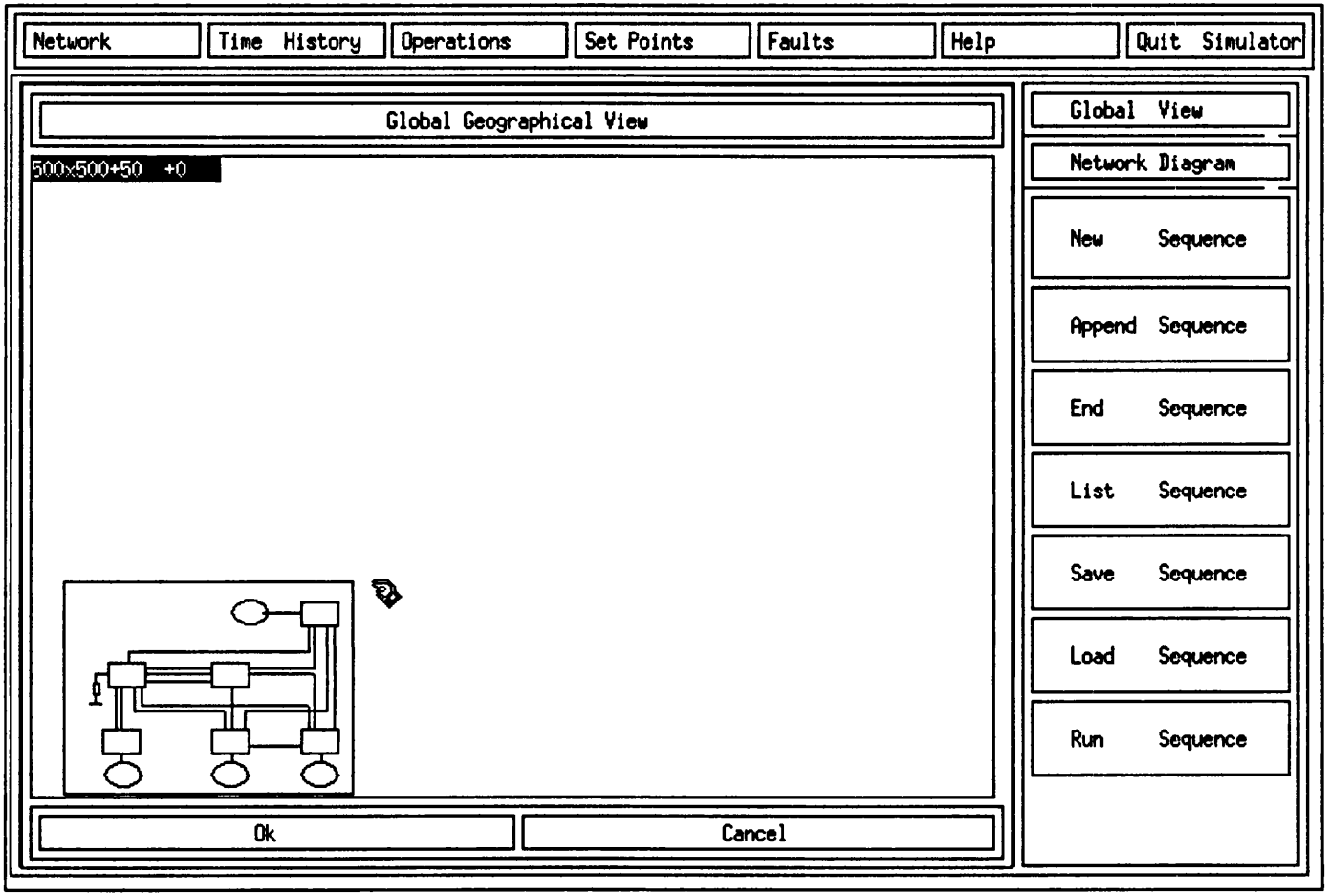


Figure 7.12: Global view of the whole network diagram

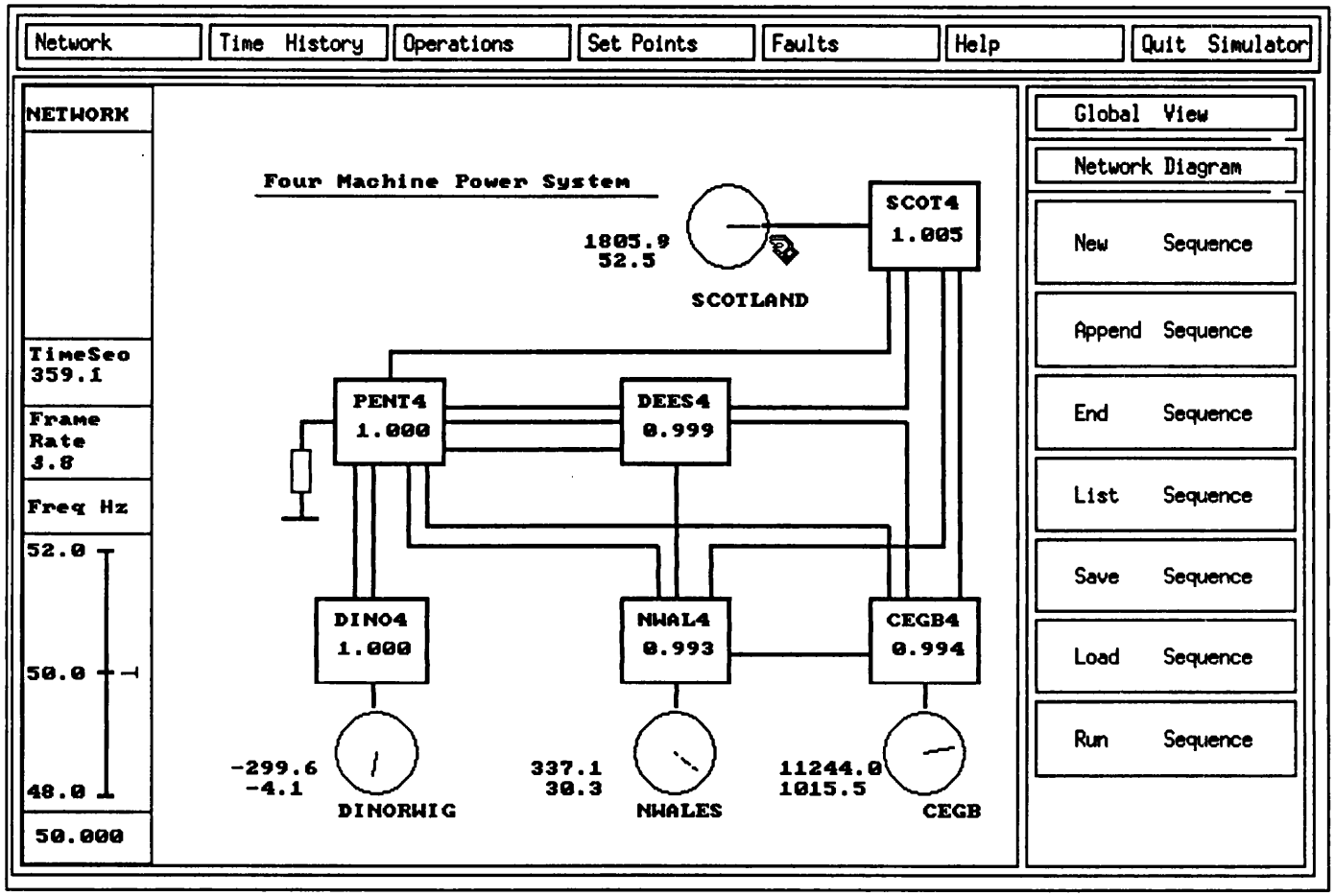


Figure 7.13: Specifying a new sequence

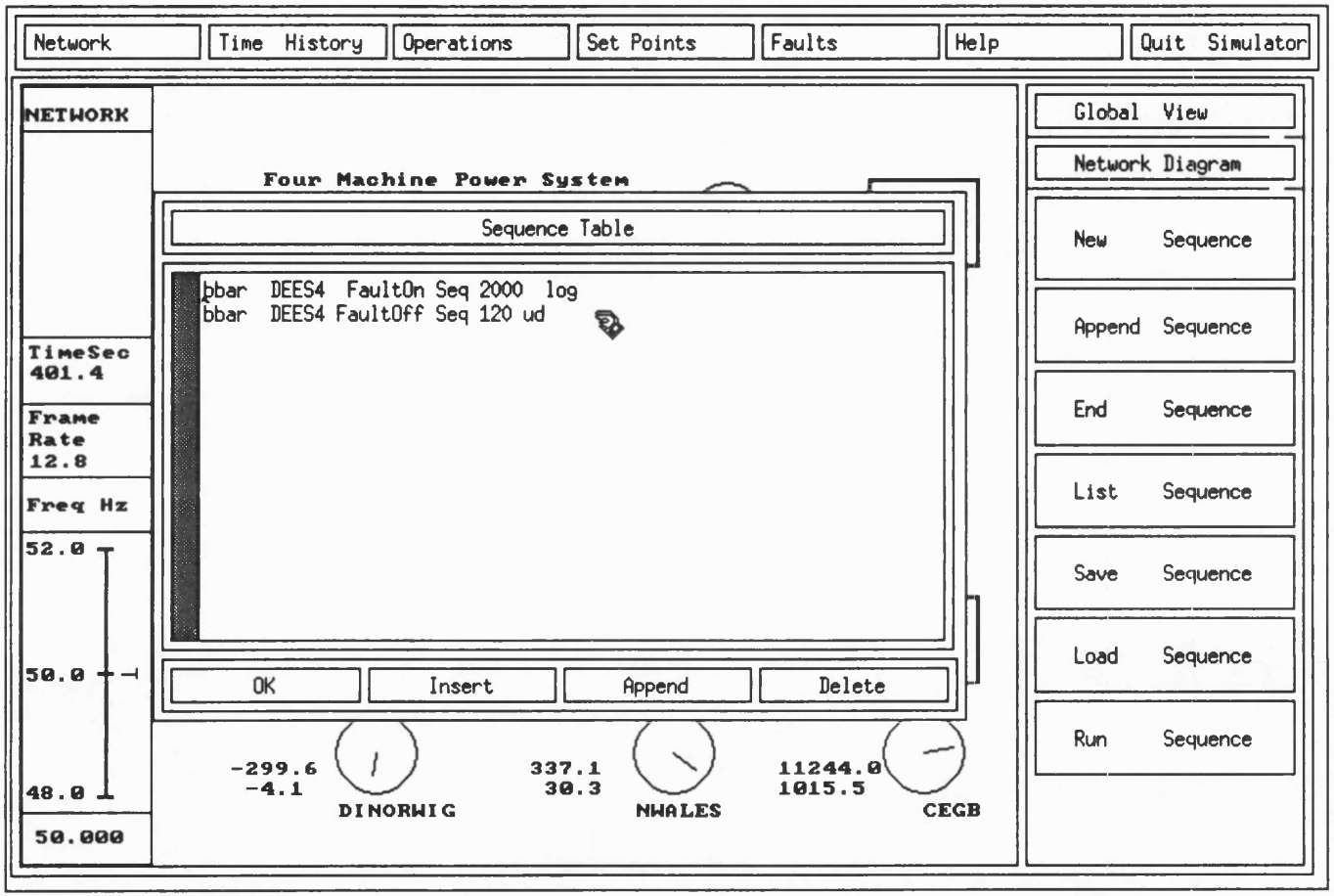


Figure 7.14: Table of content of a newly constructed sequence

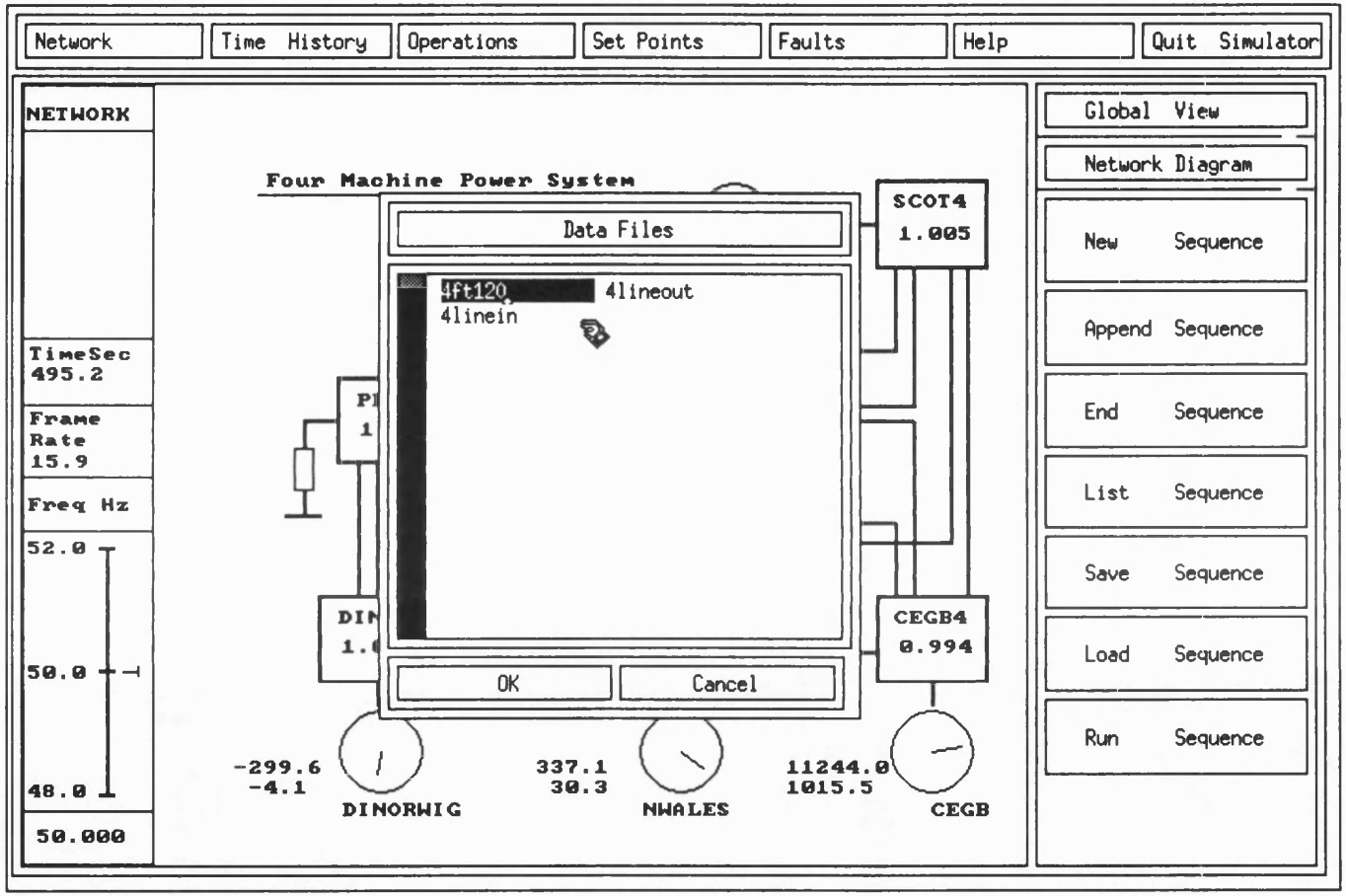


Figure 7.15: Selecting a sequence file

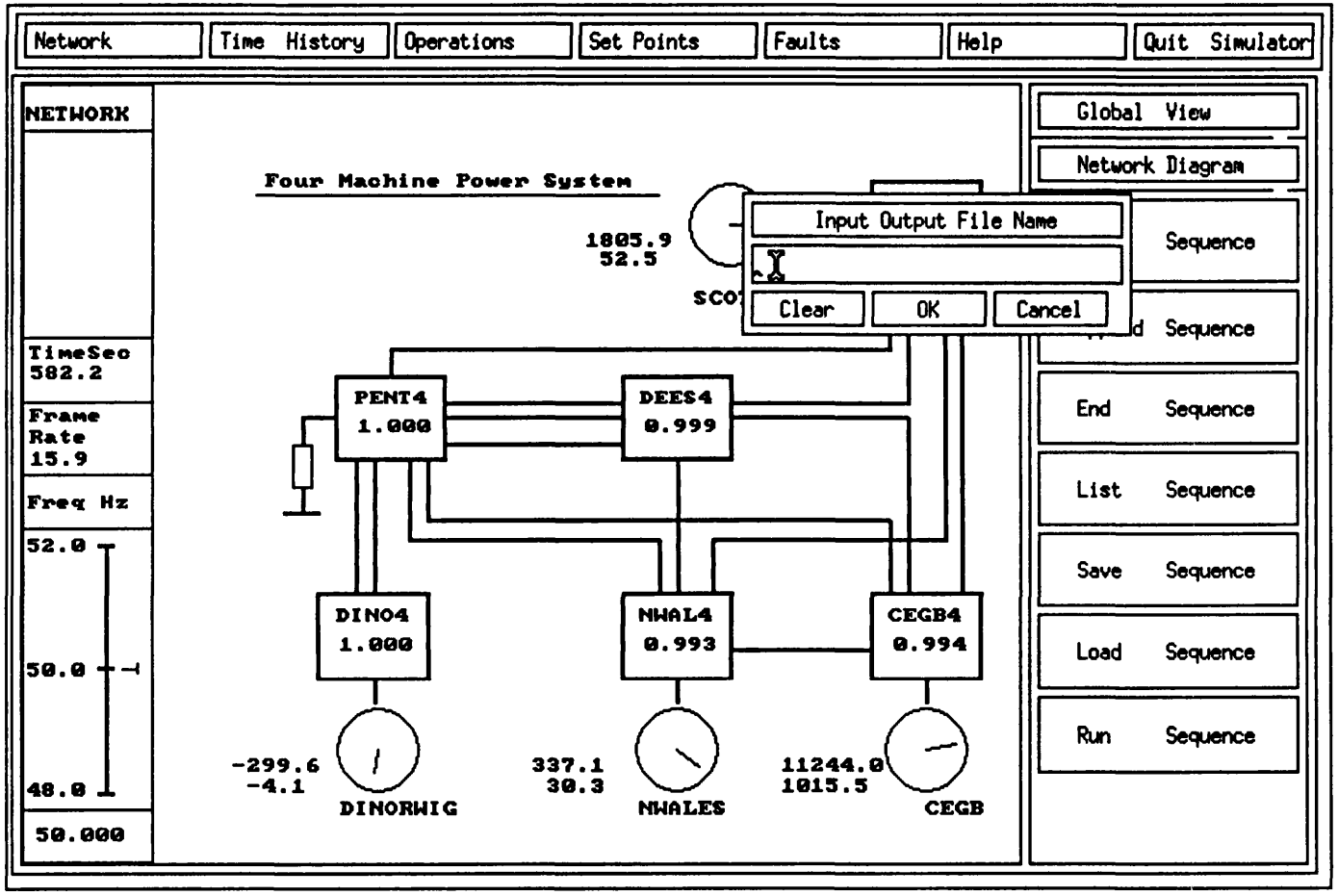


Figure 7.16: Specifying the ouput file name for a stored sequence

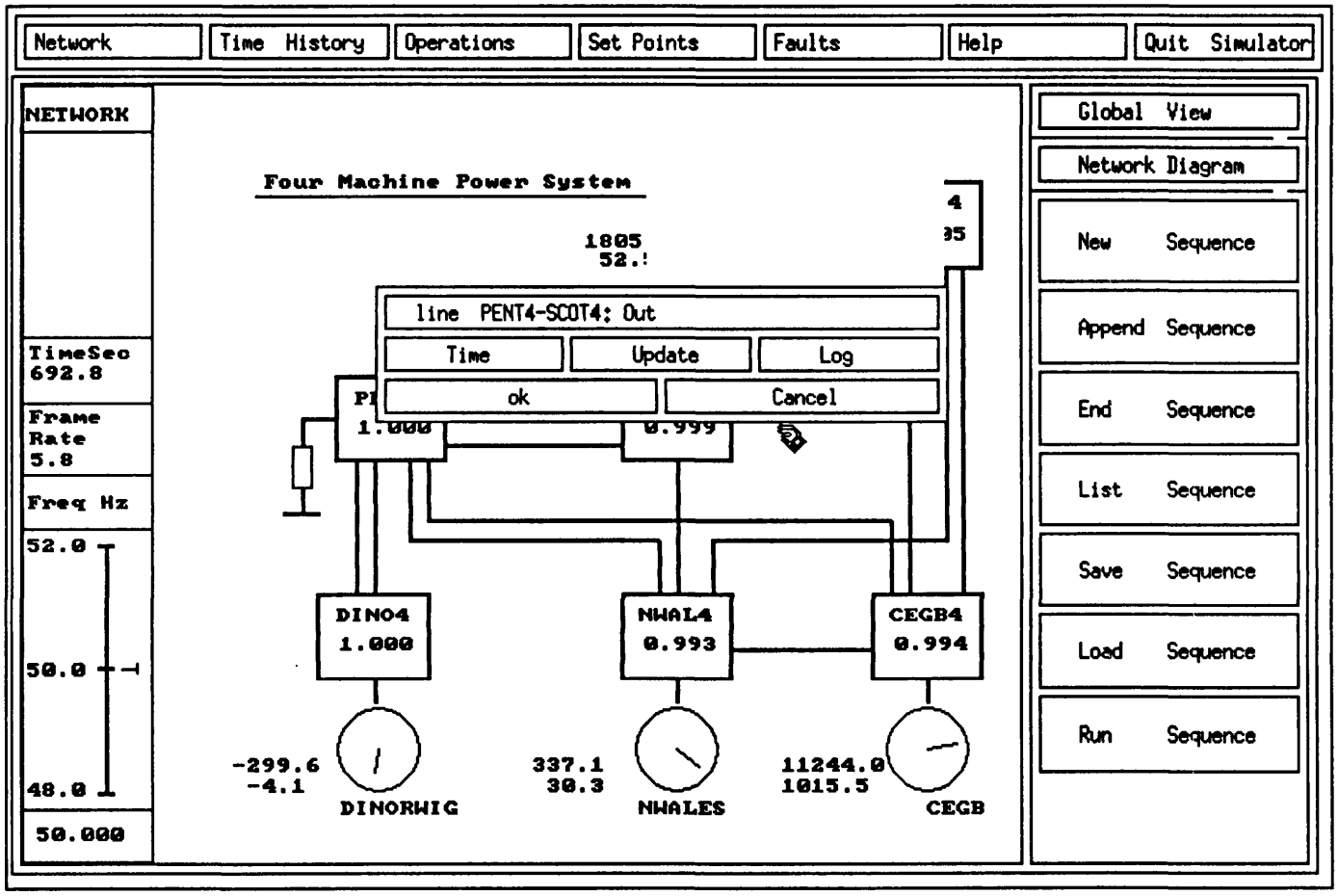


Figure 7.17: Specifying the detail of an event

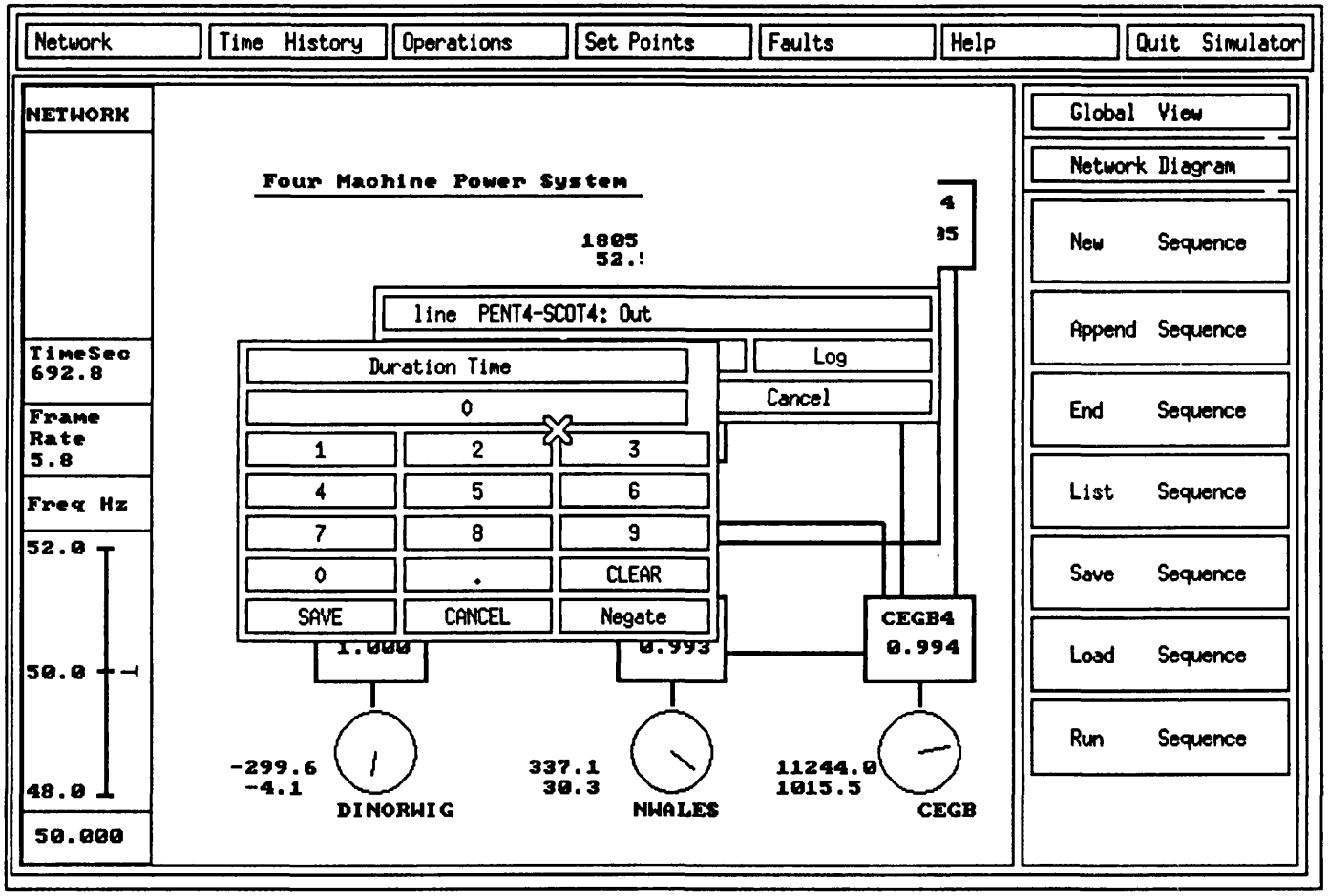


Figure 7.18: Specifying the time duration of an event

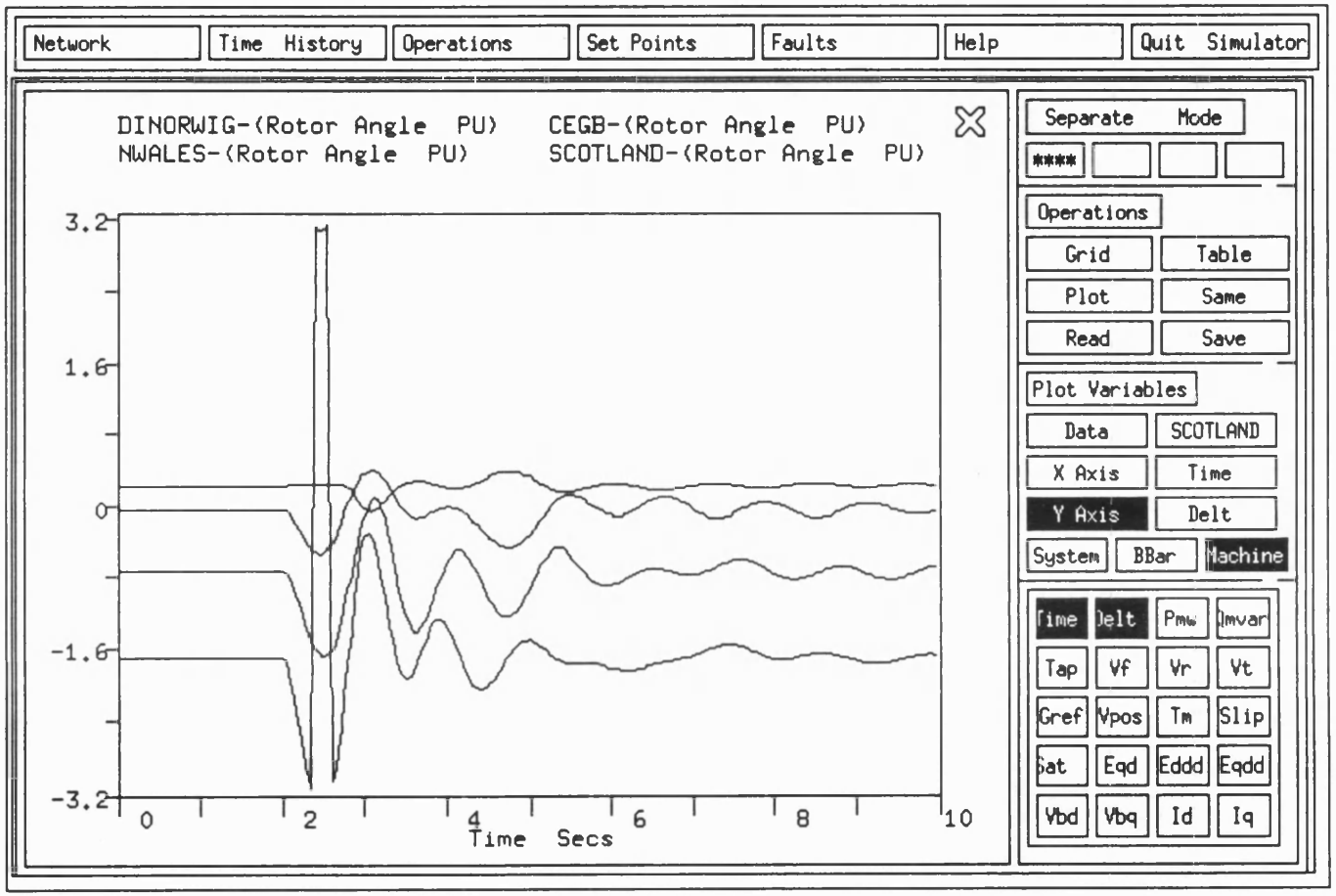


Figure 7.19: Four superimposed graphs

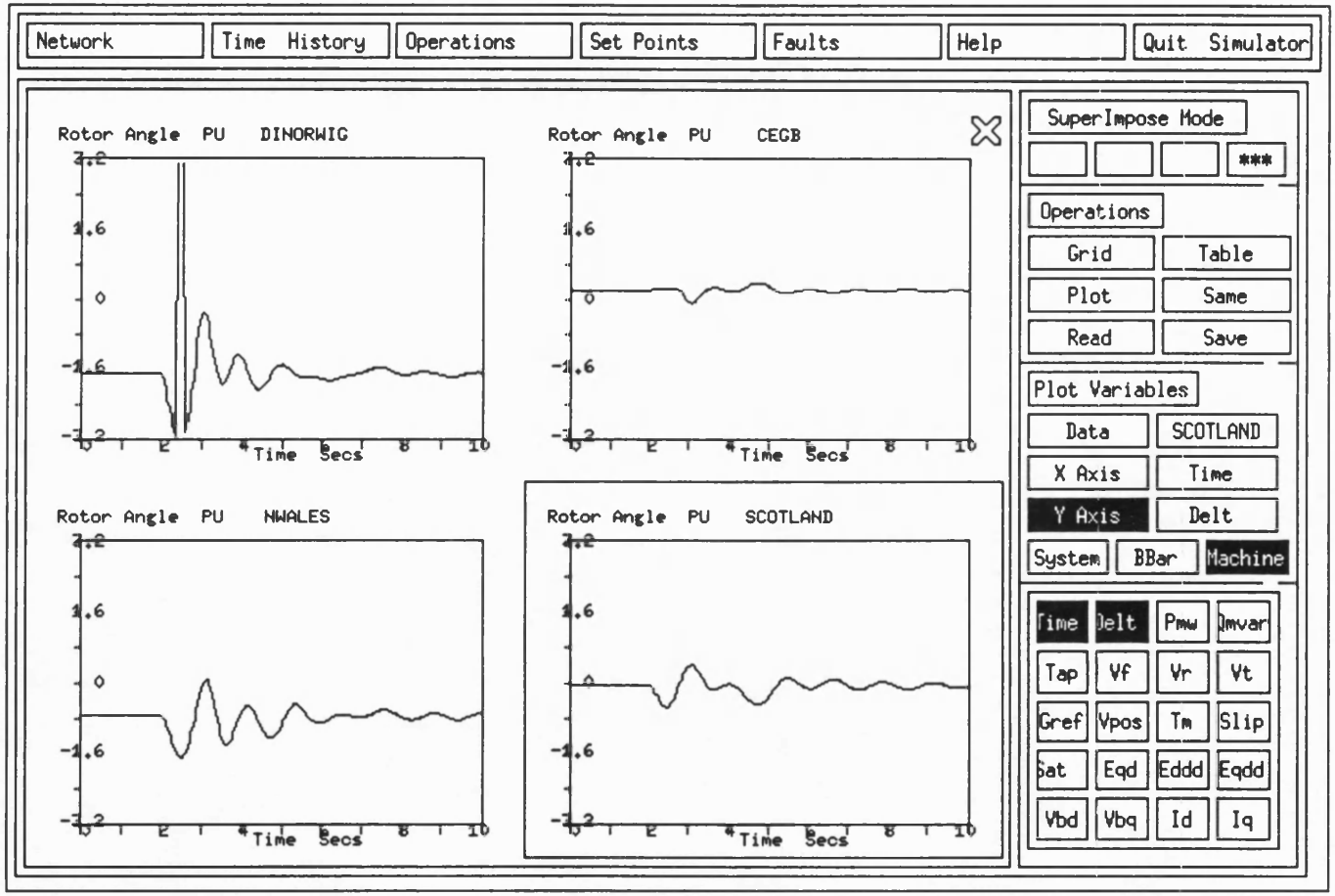


Figure 7.20: Four separately displayed graphs

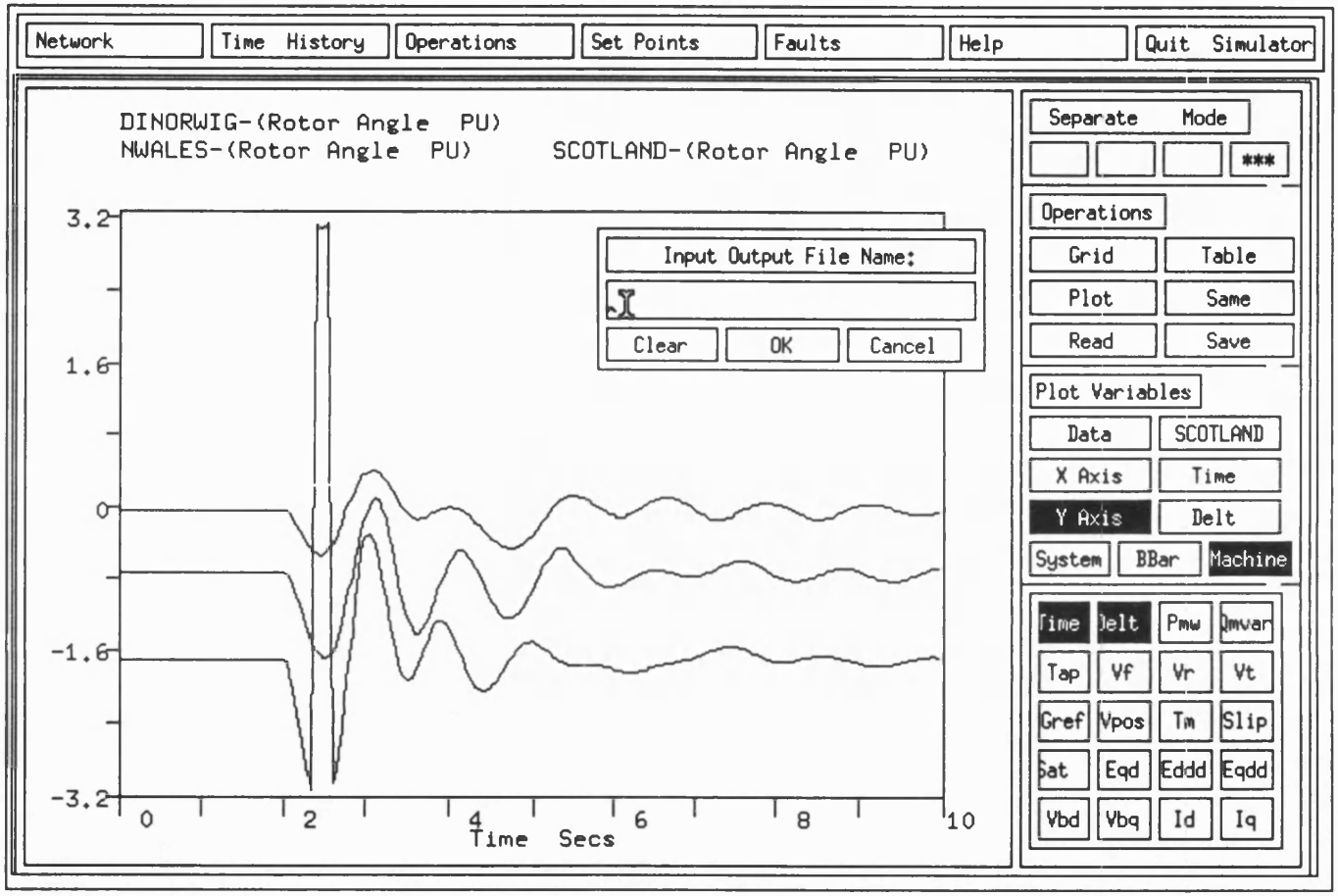


Figure 7.21: Specifying the output file name for the data of a graph

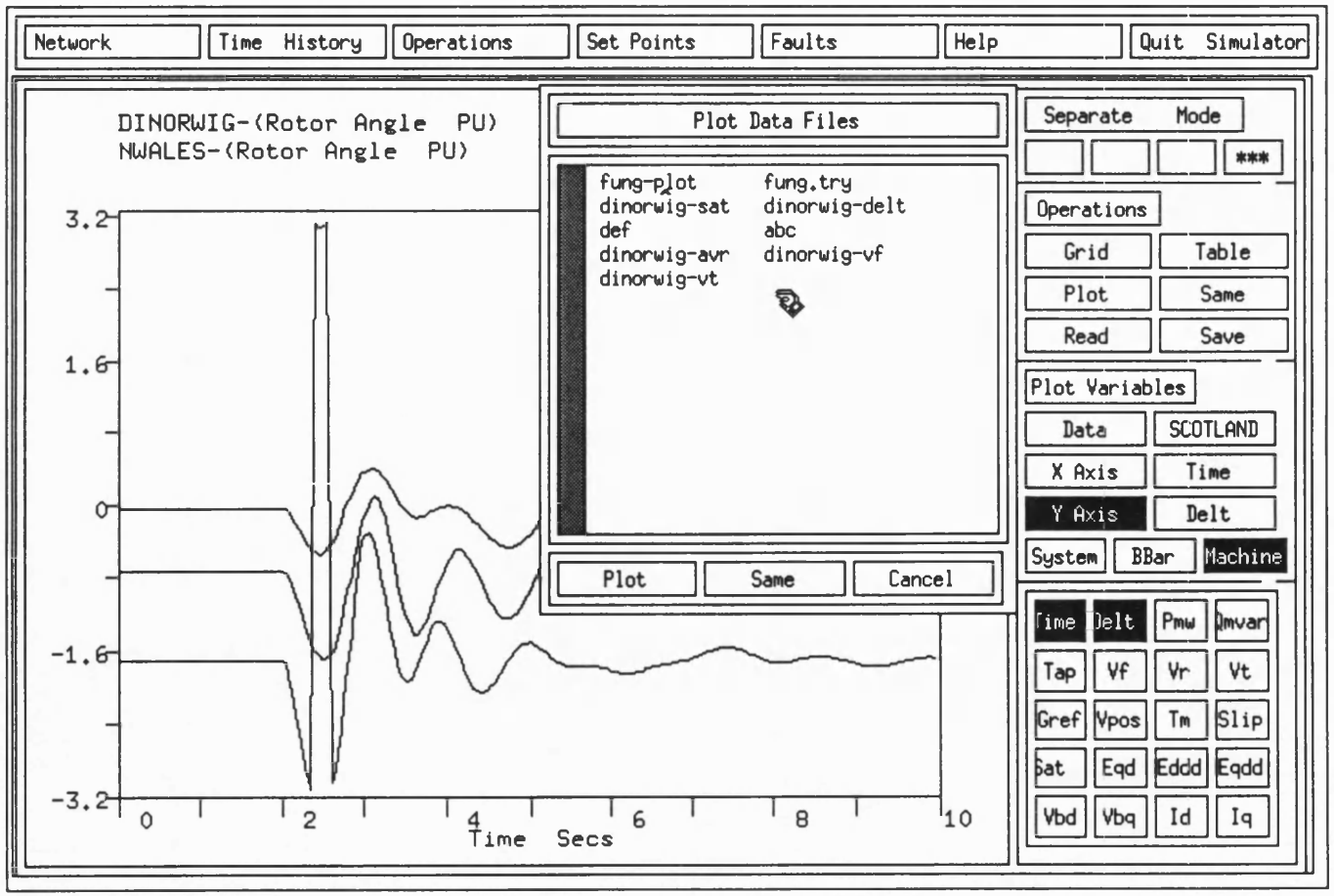


Figure 7.22: Selecting an input data file

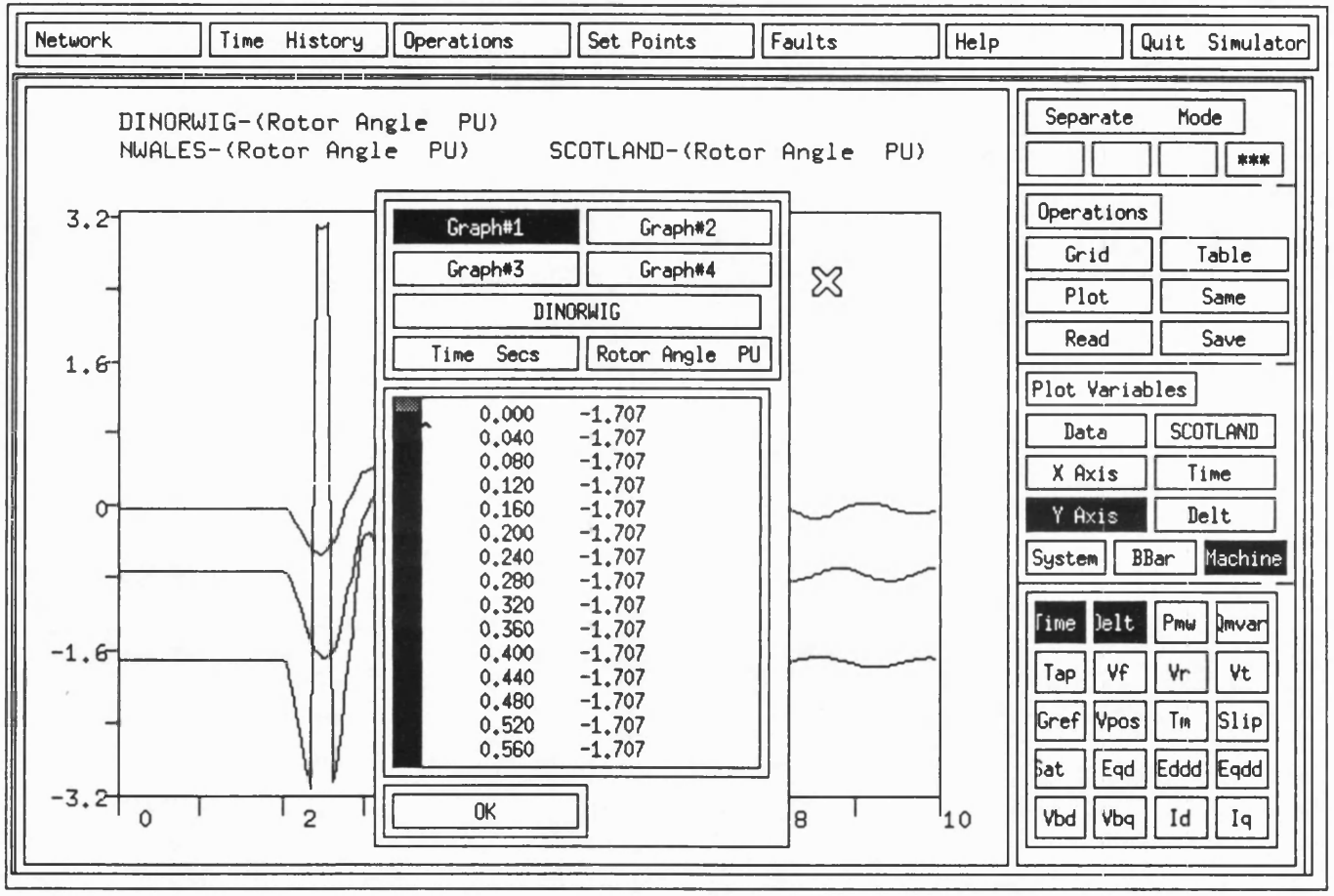


Figure 7.23: Table diaplaying the numerical data of a graph

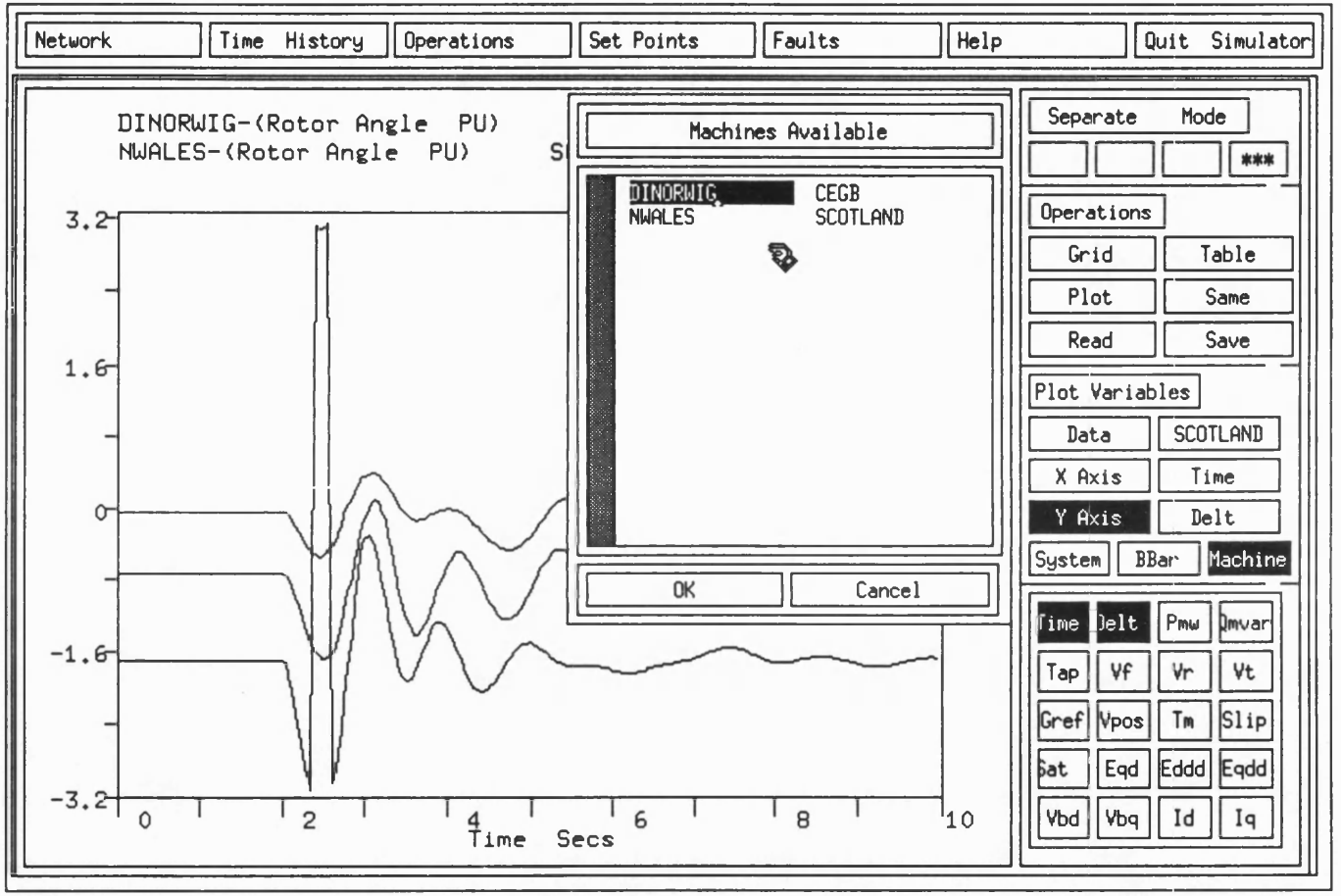


Figure 7.24: Selecting a machine as the new data source for graph plotting

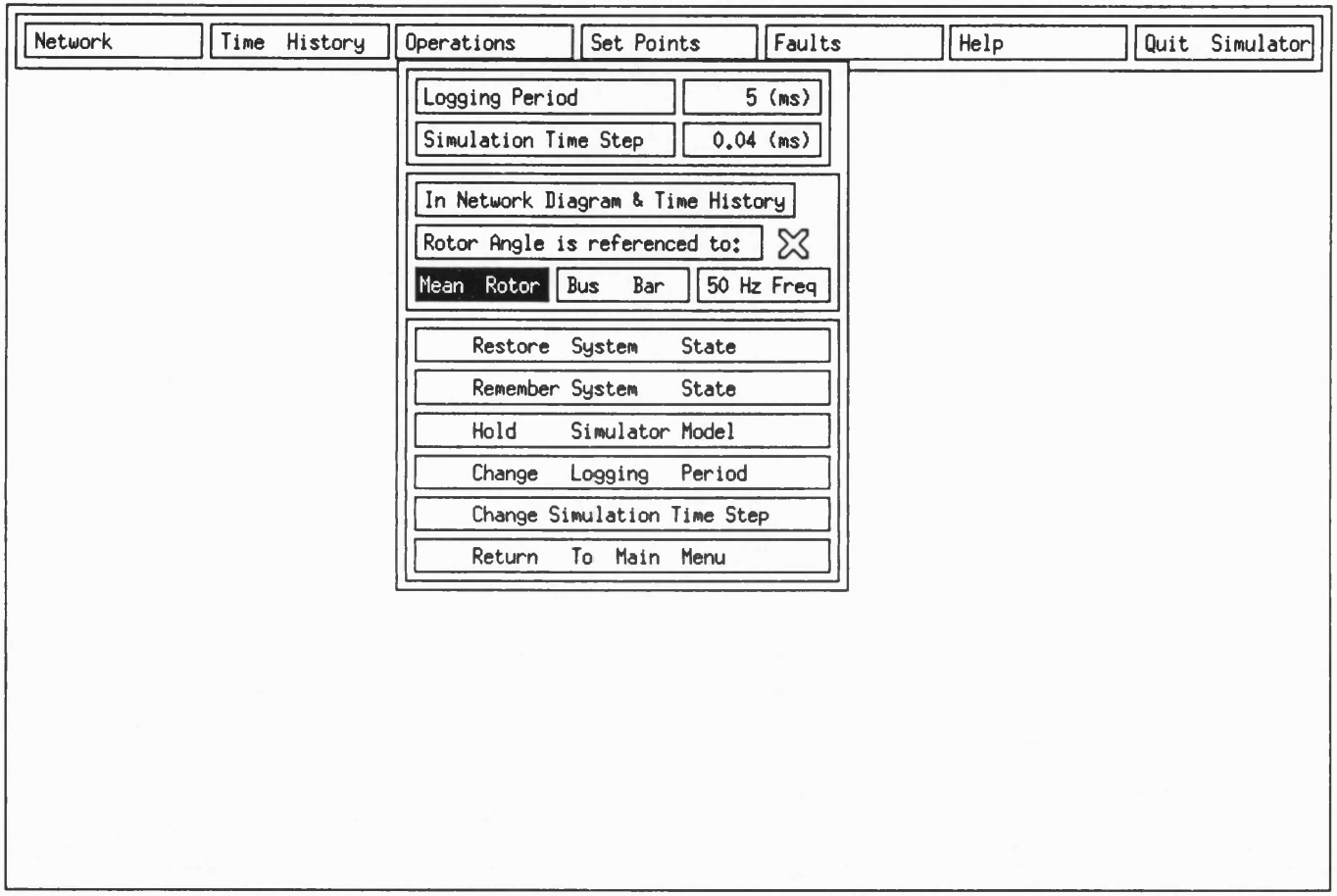


Figure 7.25: The Operation Menu

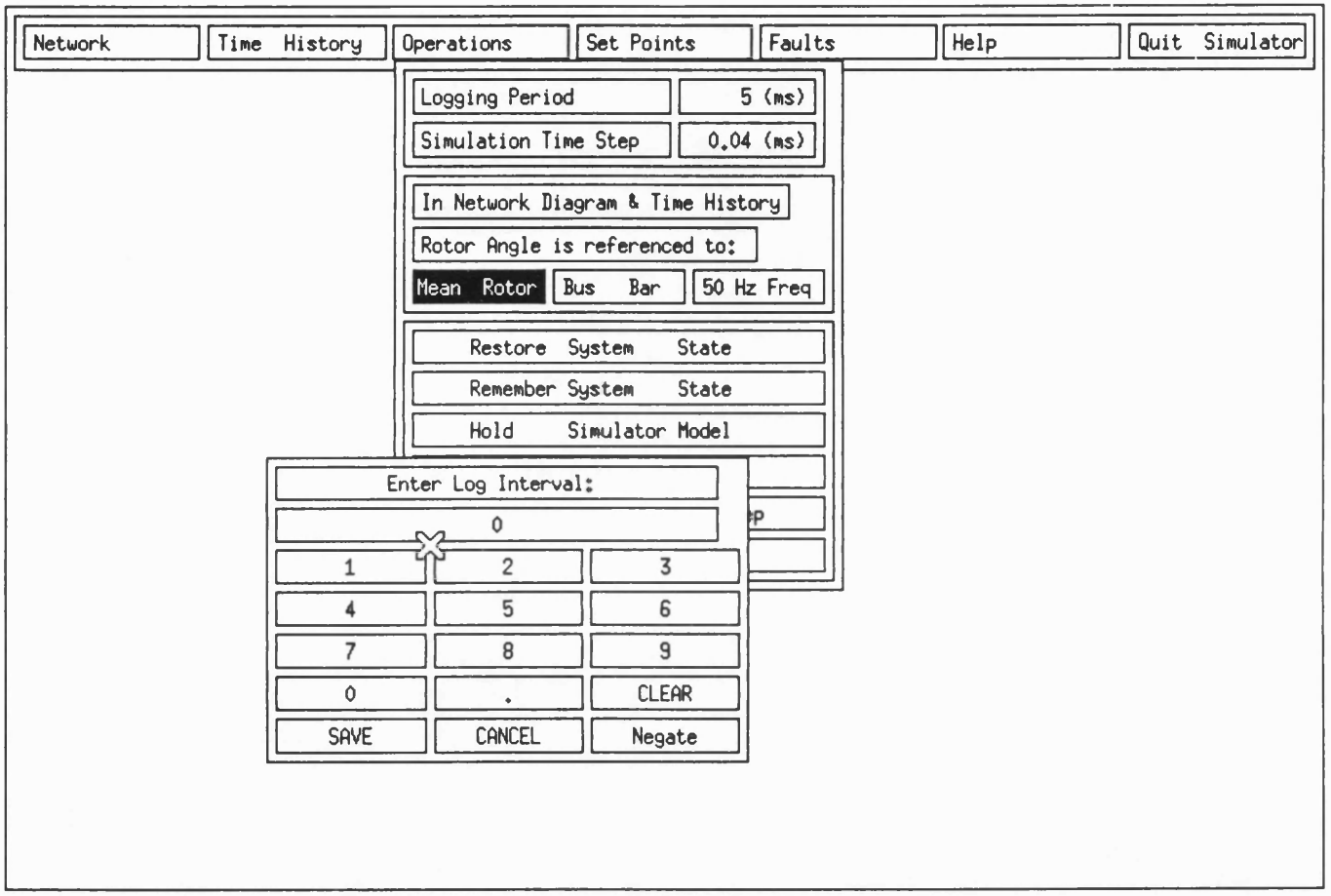


Figure 7.26: Changing the logging period

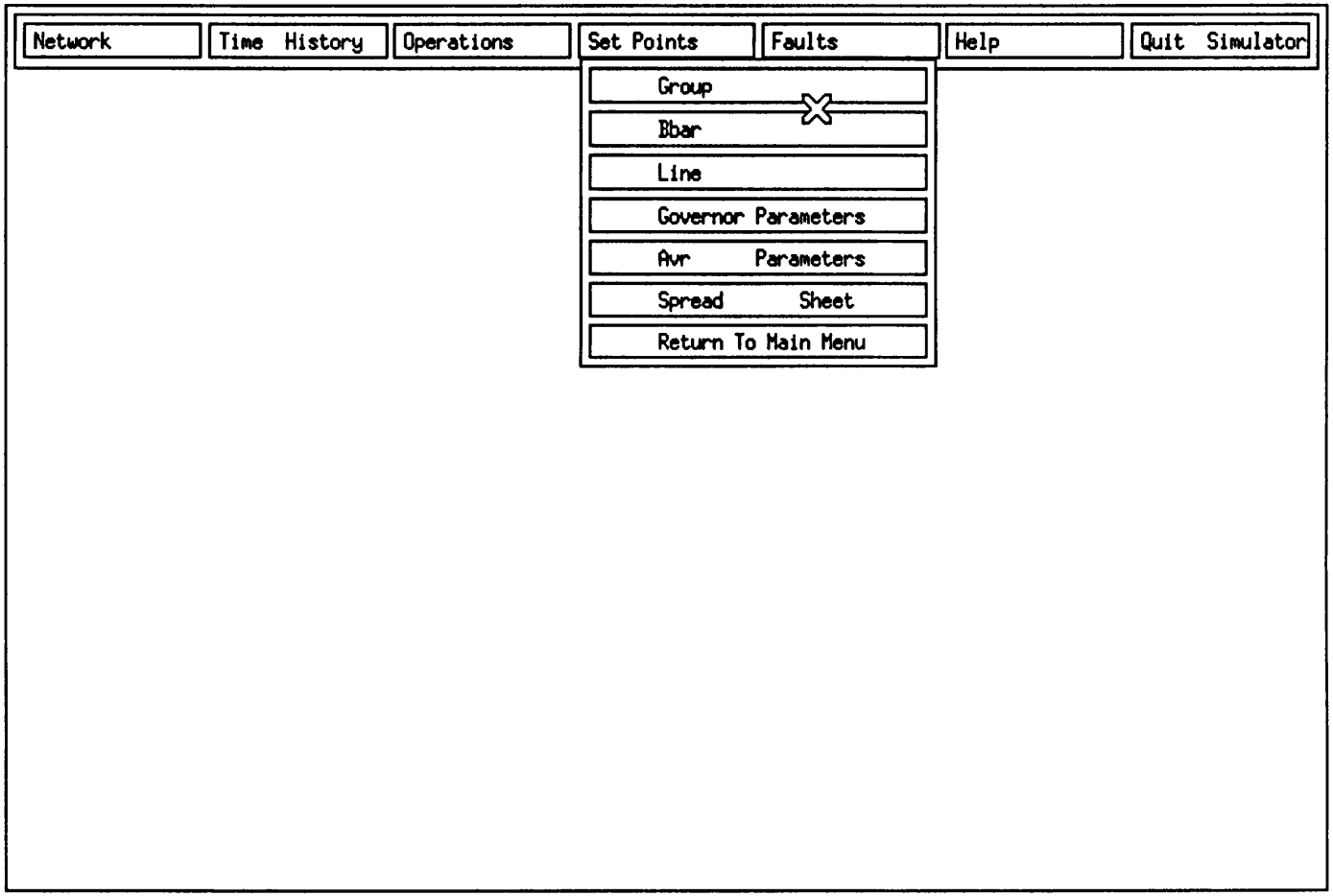


Figure 7.27: Set points menu

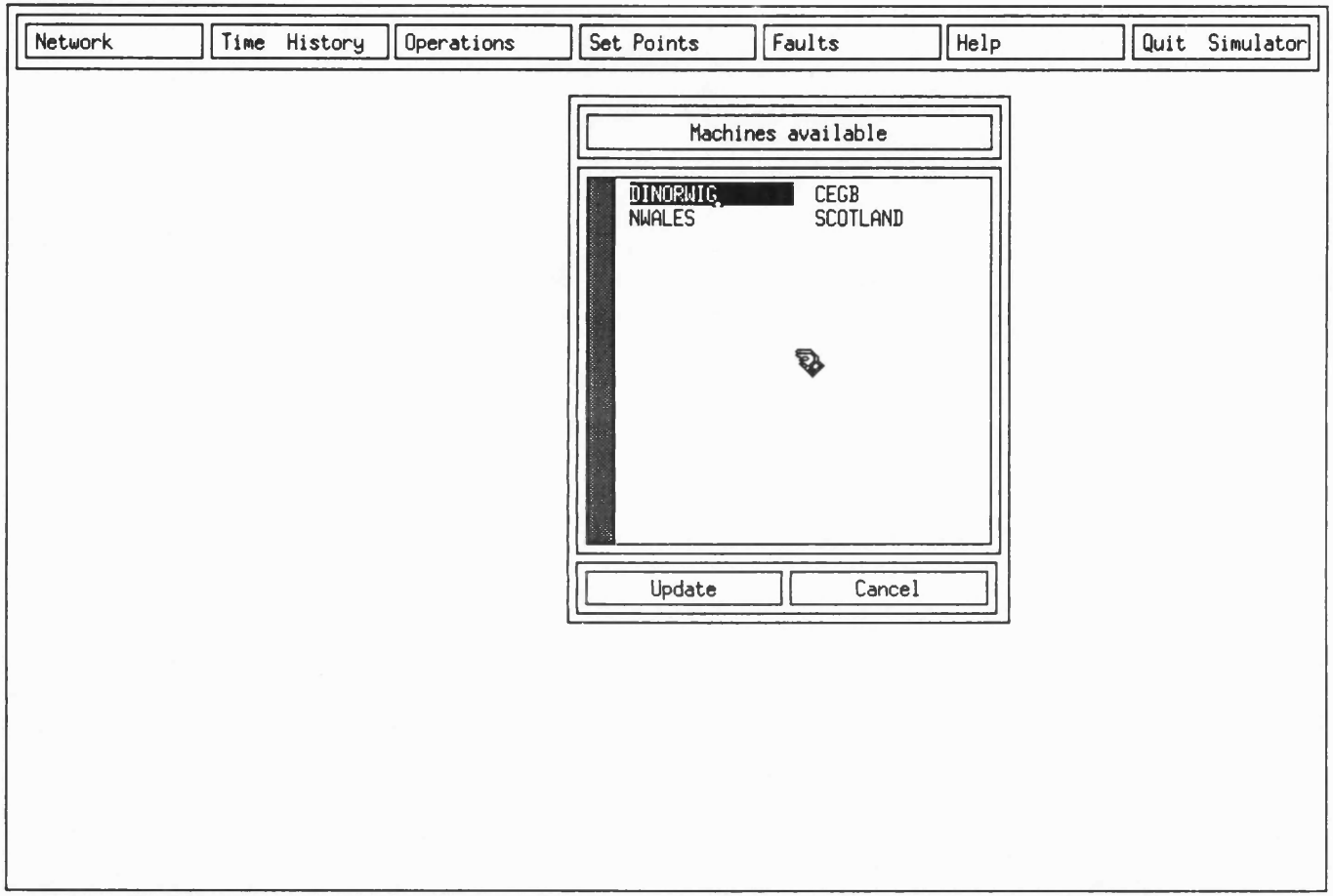


Figure 7.28: Selecting a machine for modification

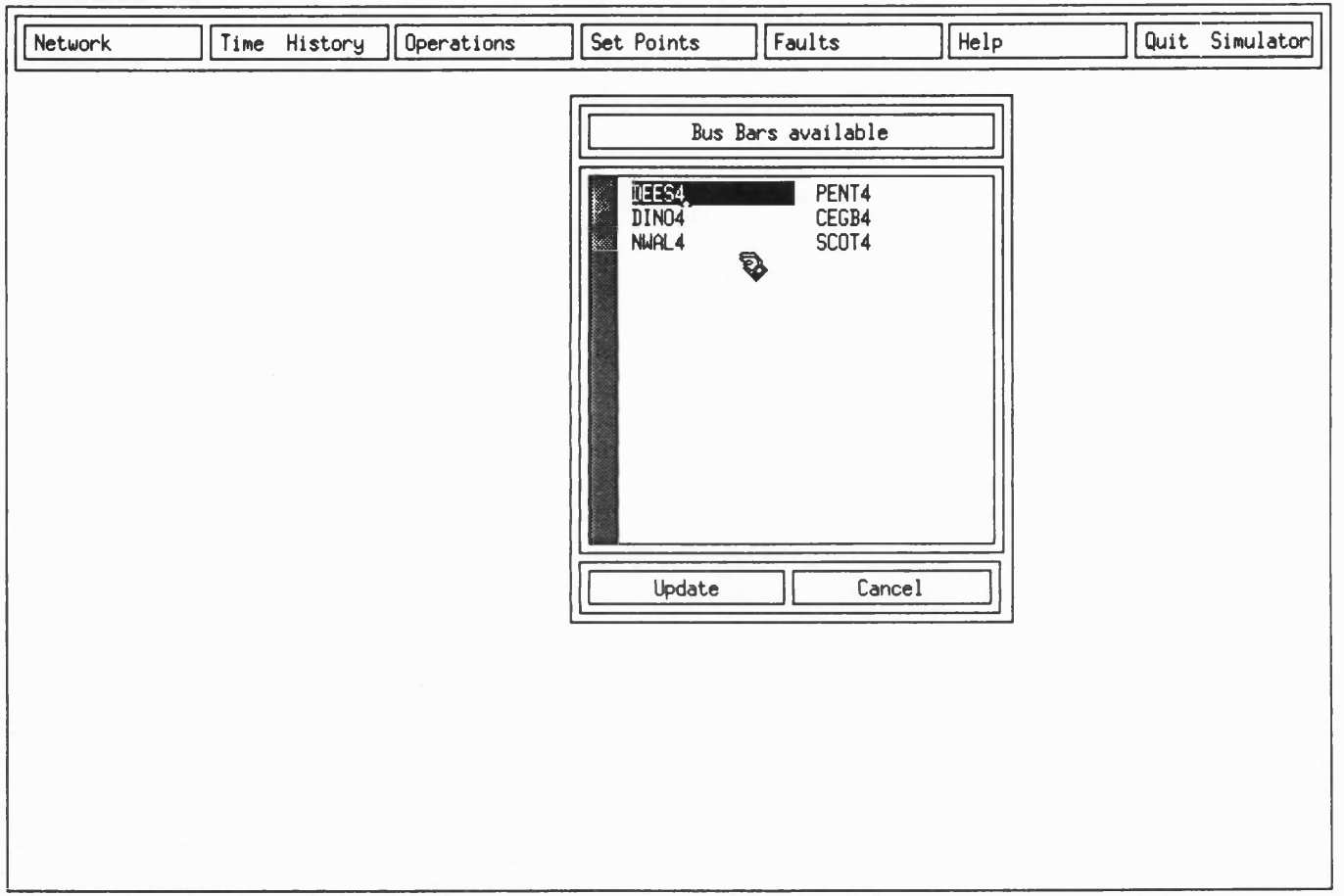


Figure 7.29: Selecting a busbar for modification

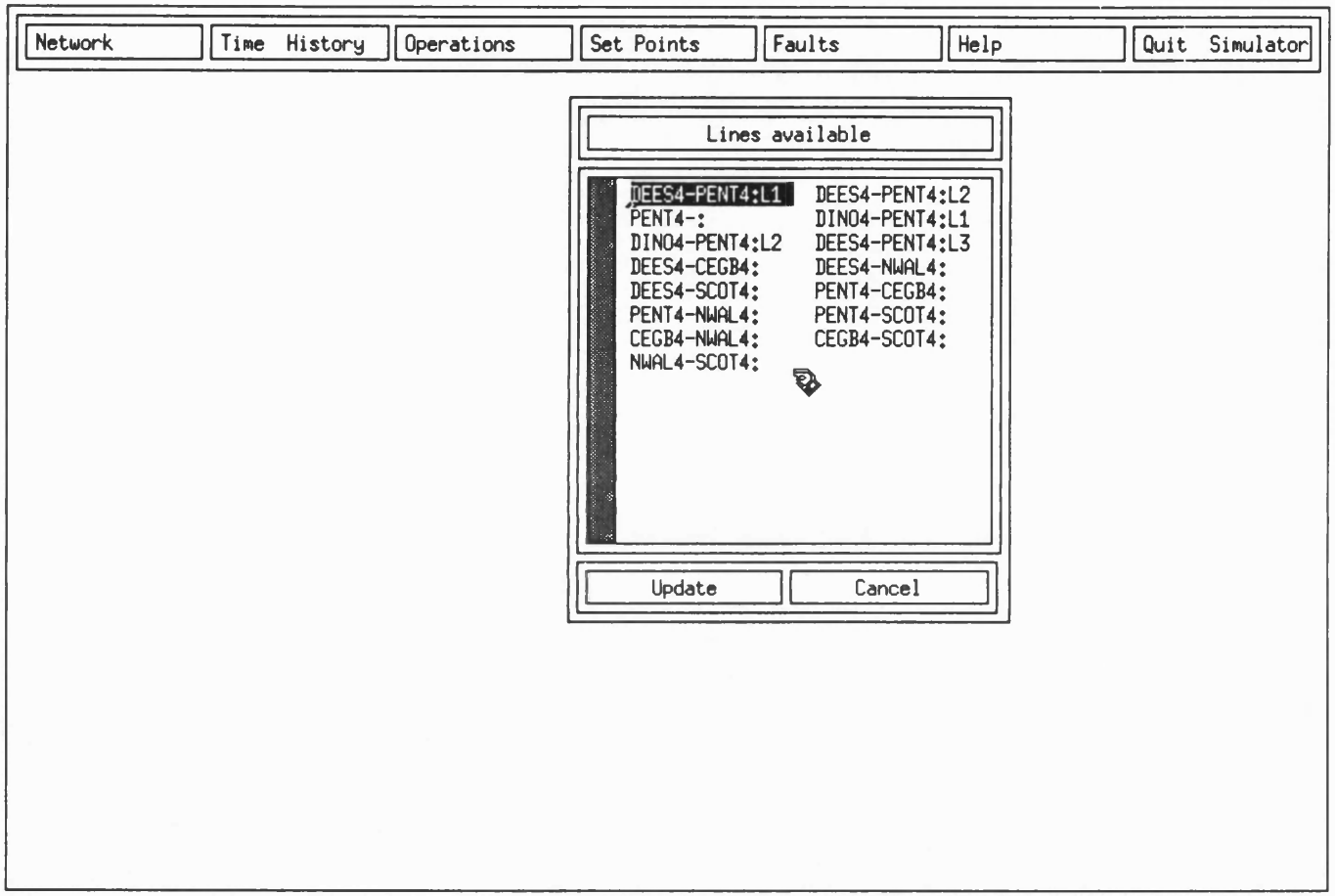


Figure 7.30: Selecting a line for modification

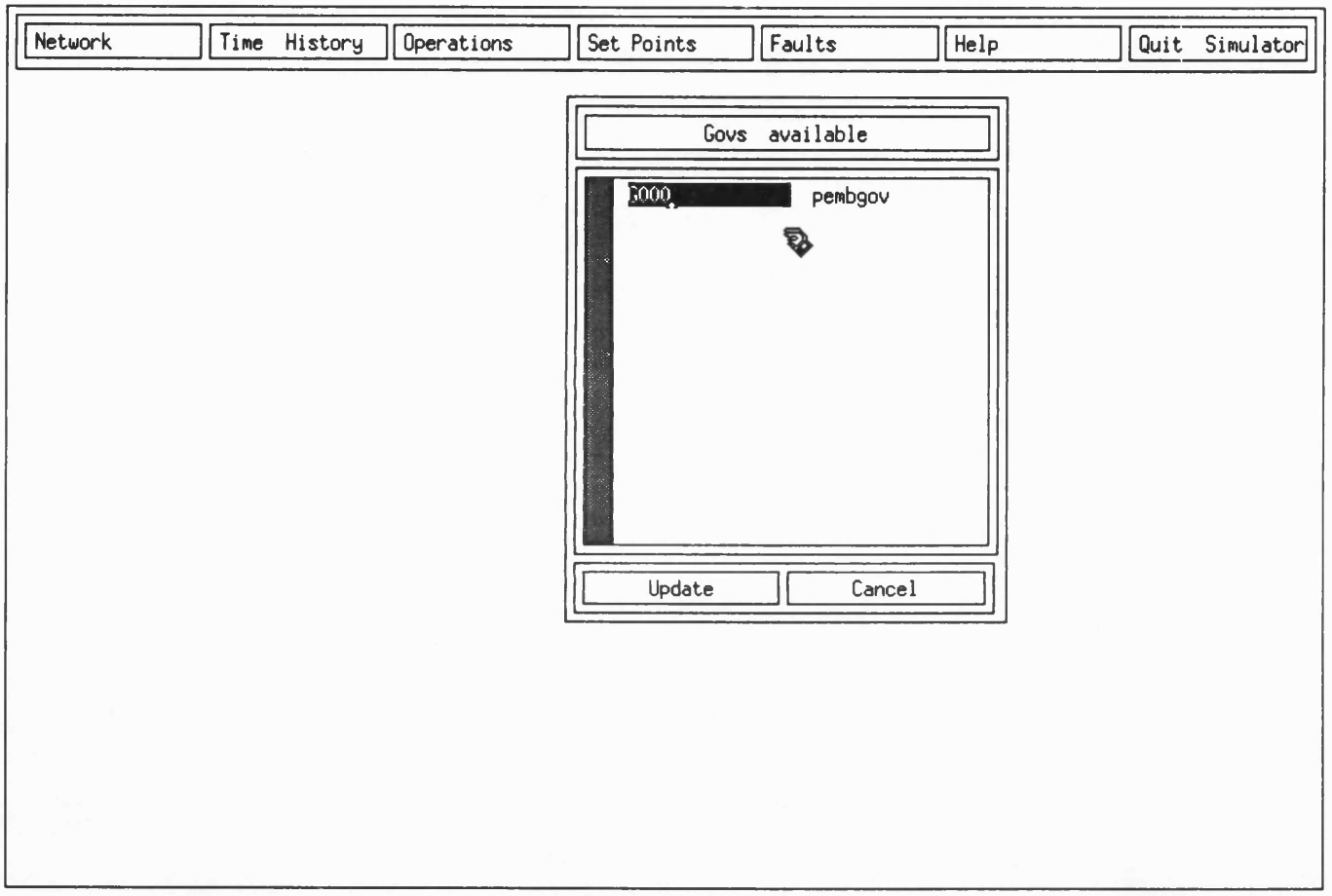


Figure 7.31: Selecting a governor for modification

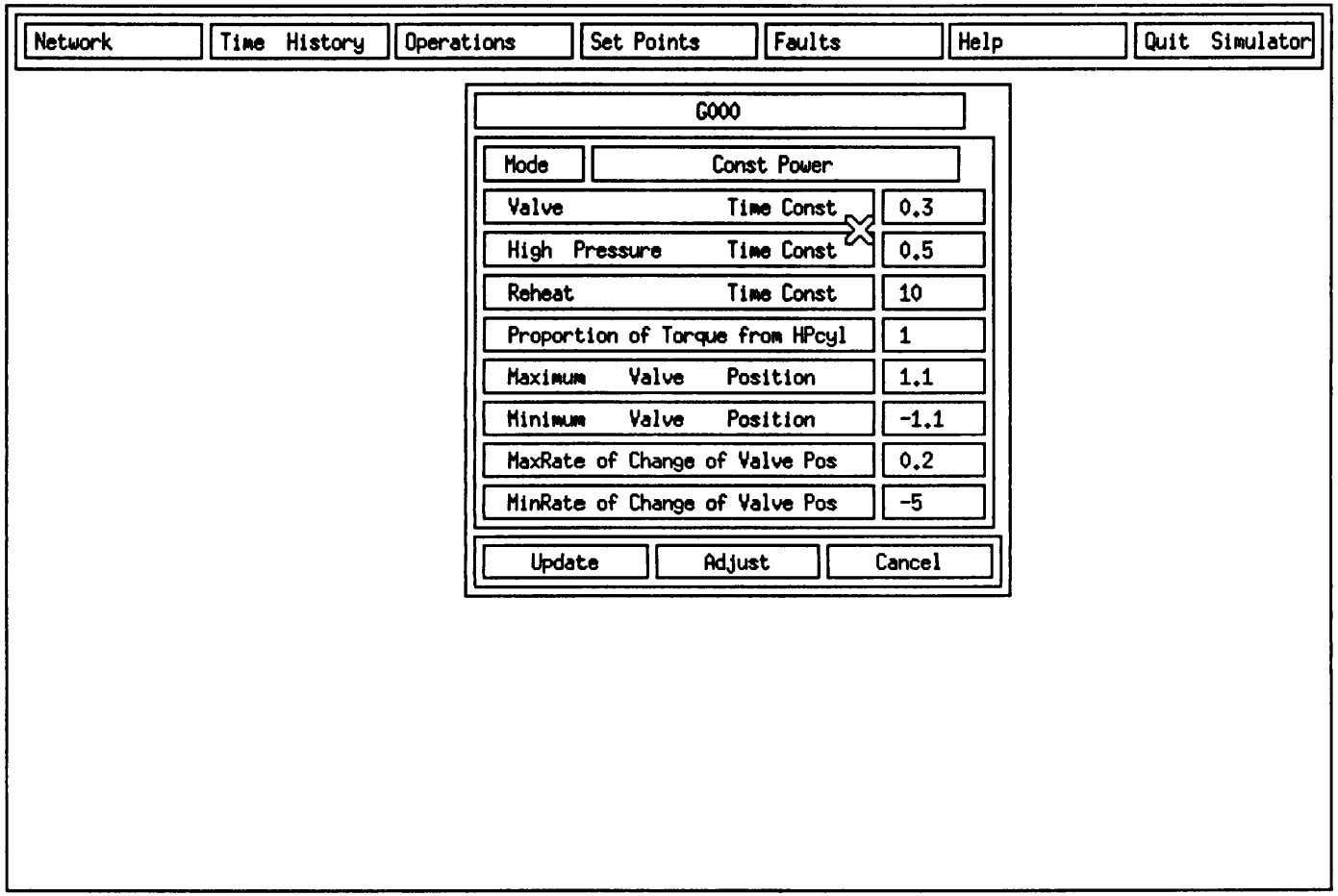


Figure 7.32: Menu of set points of a governor

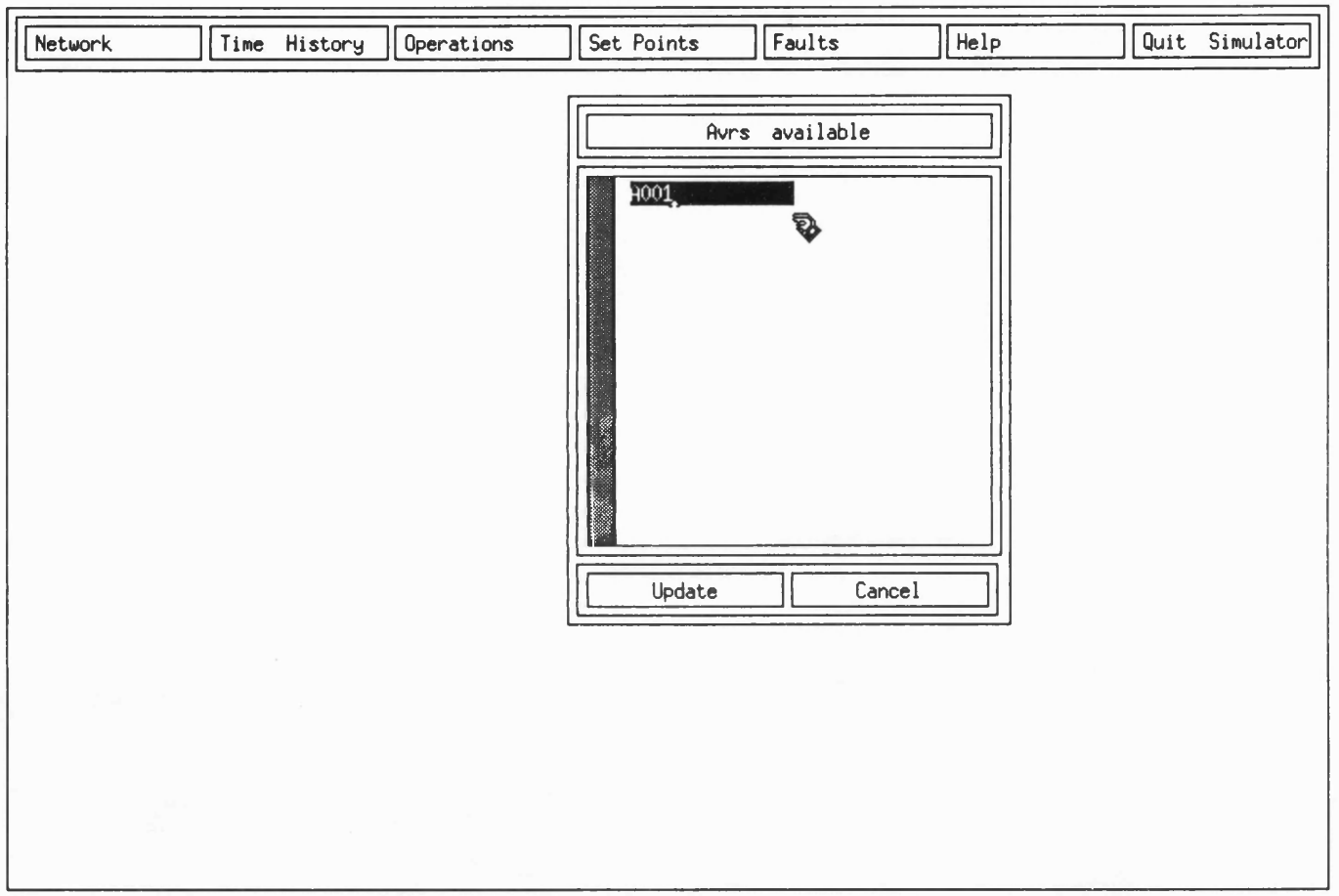


Figure 7.33: Selecting an avr for modification

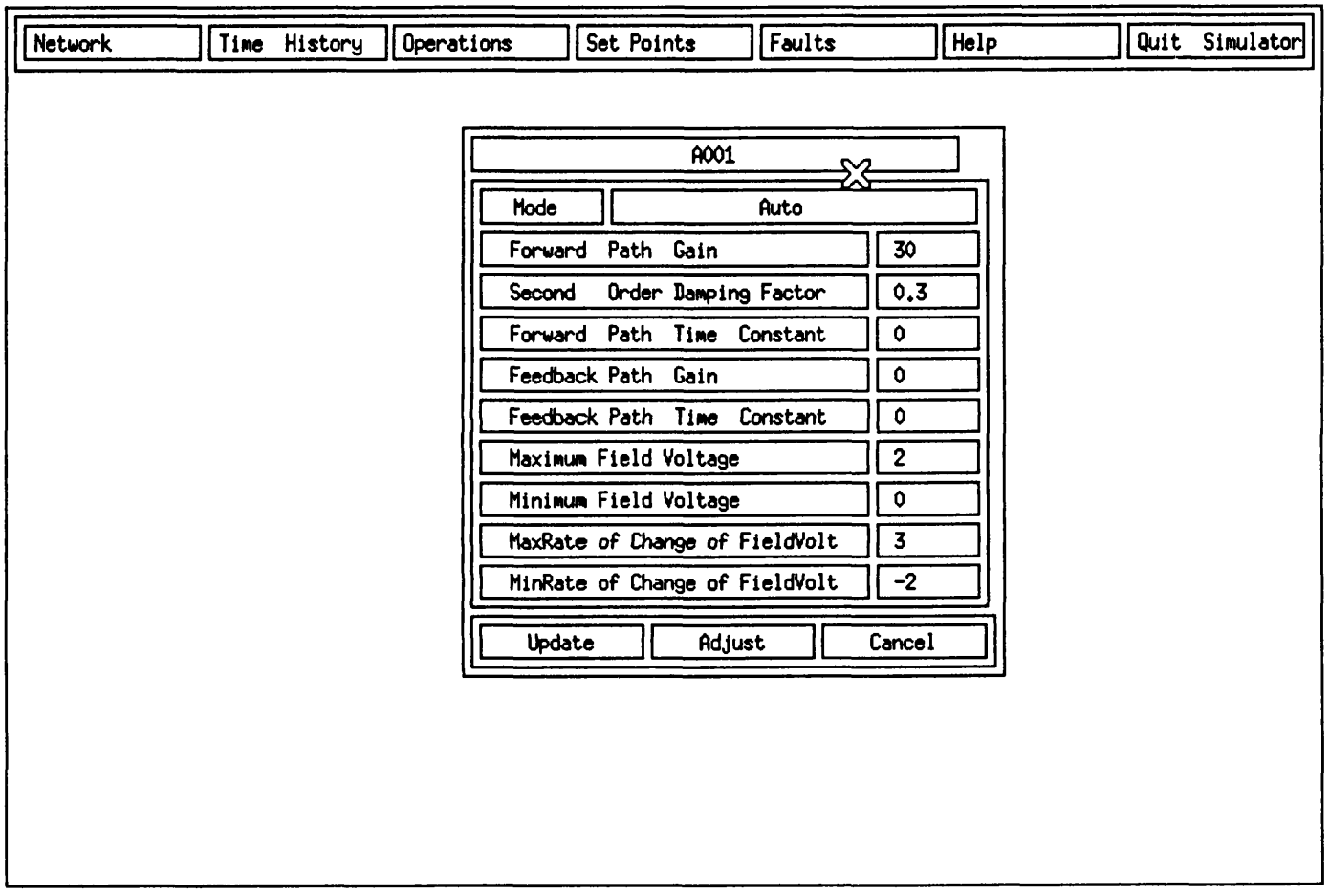


Figure 7.34: Menu of set points of an avr

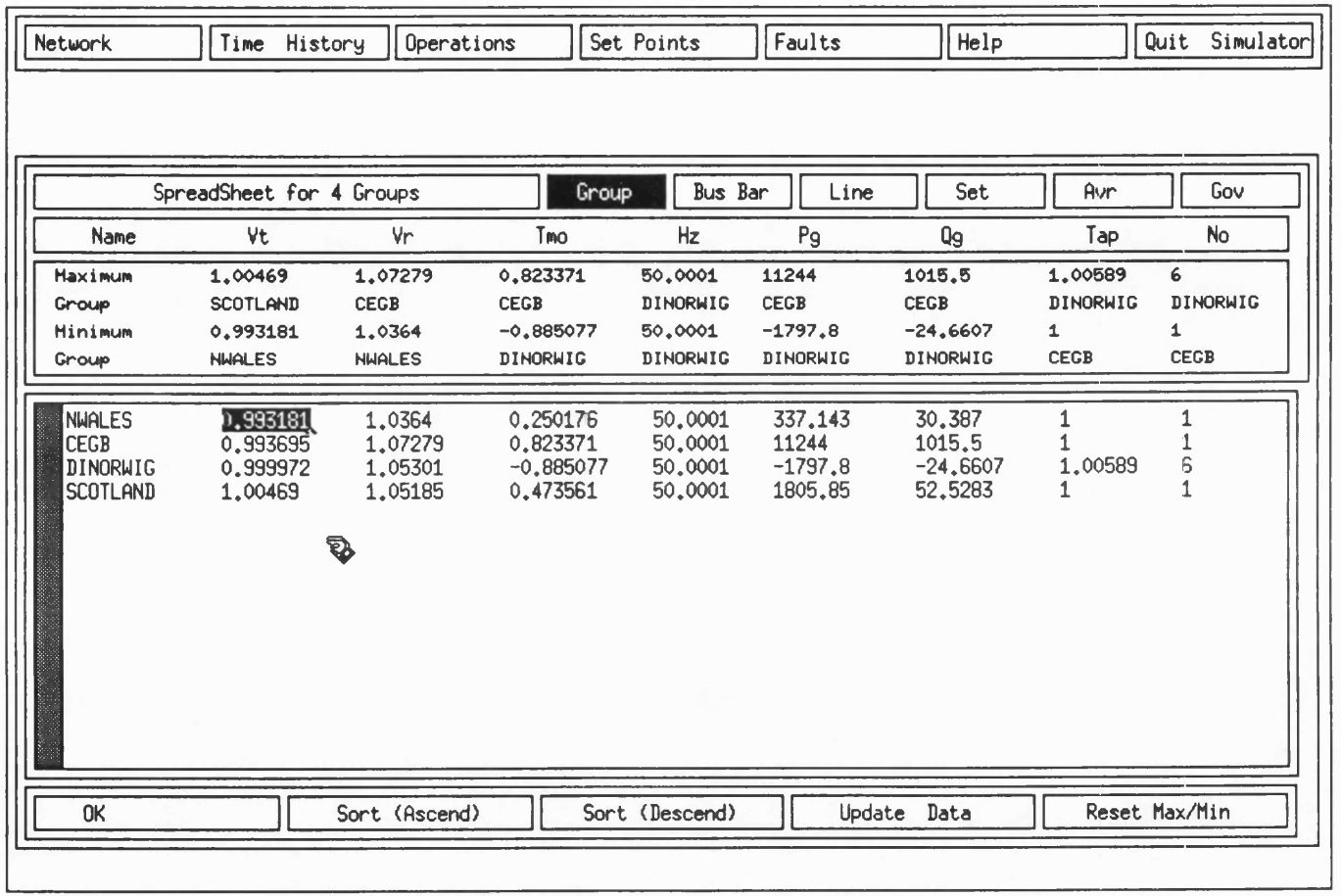


Figure 7.35: Spreadsheet of all the available machines

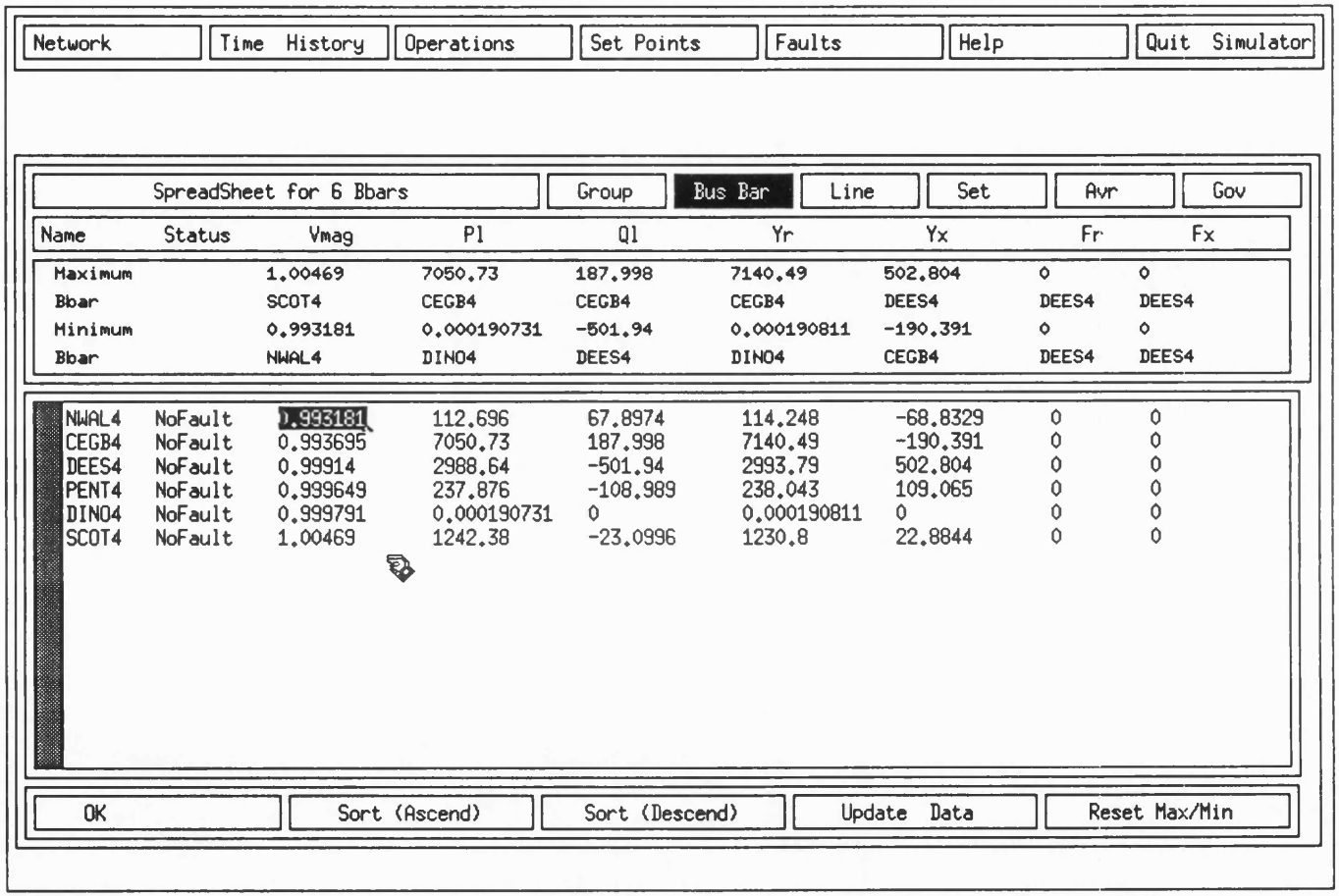


Figure 7.36: Spreadsheet of all the available busbars

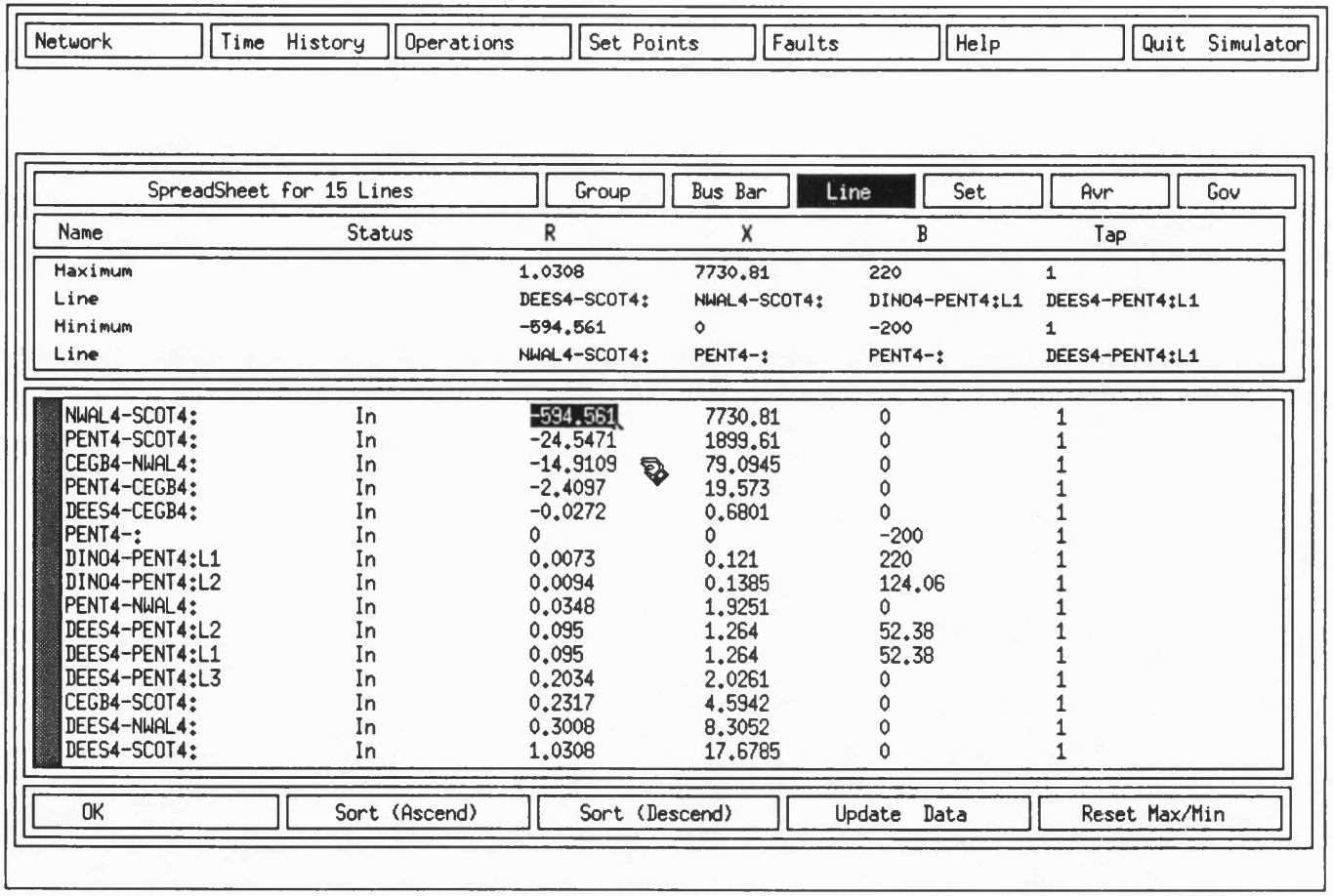


Figure 7.37: Spreadsheet of all the available lines

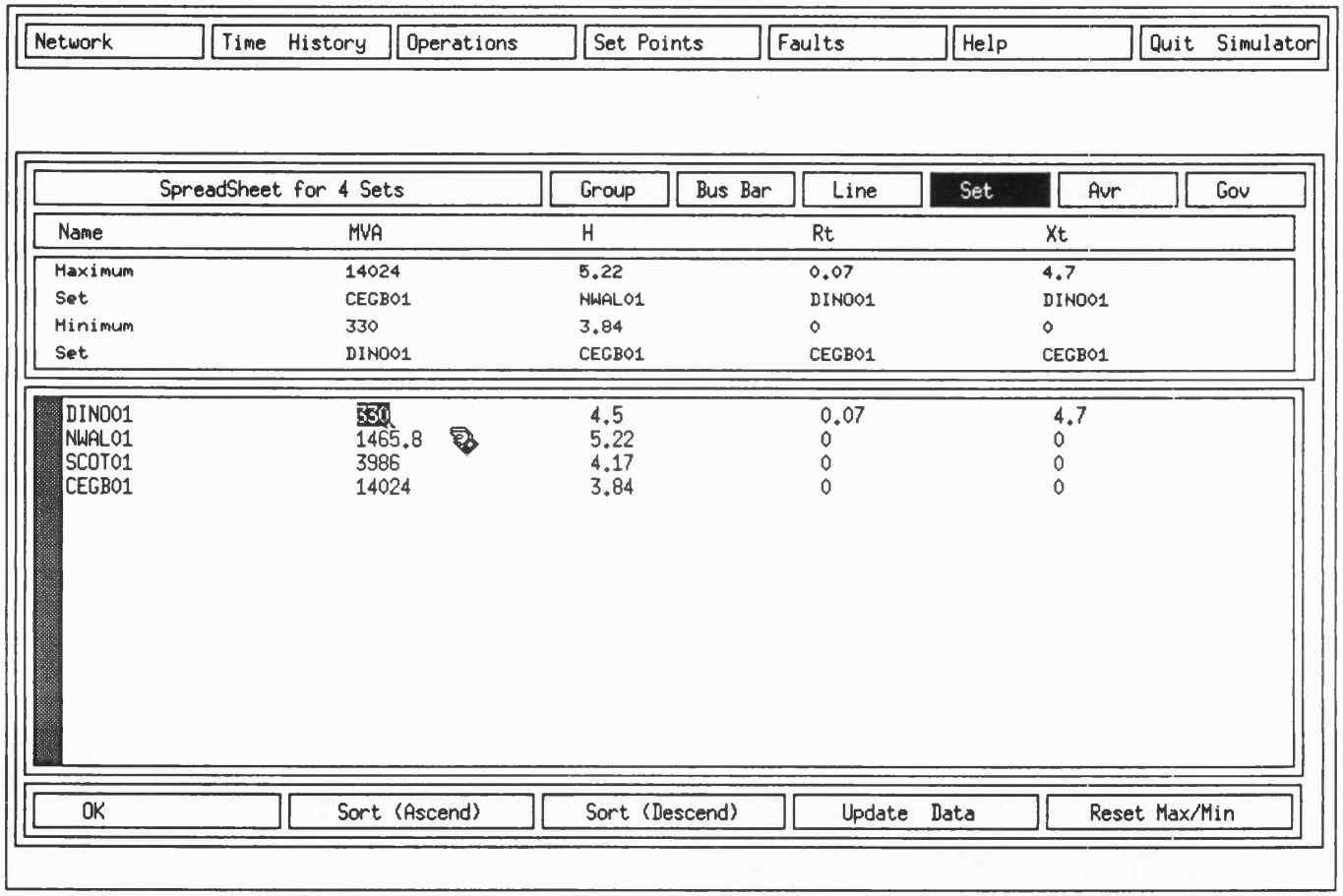


Figure 7.38: Spreadsheet of all the available sets

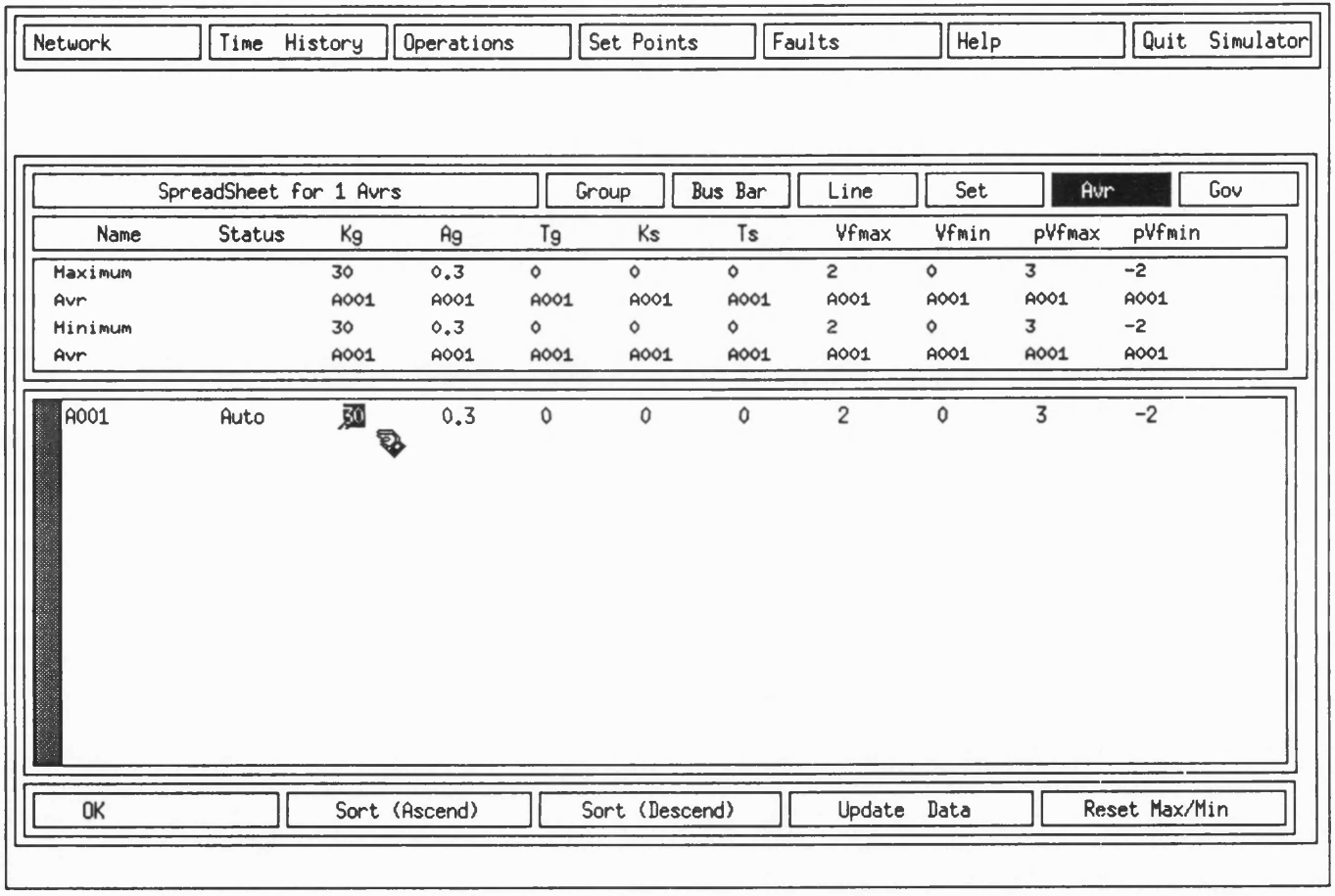


Figure 7.39: Spreadsheet of all the available avrs

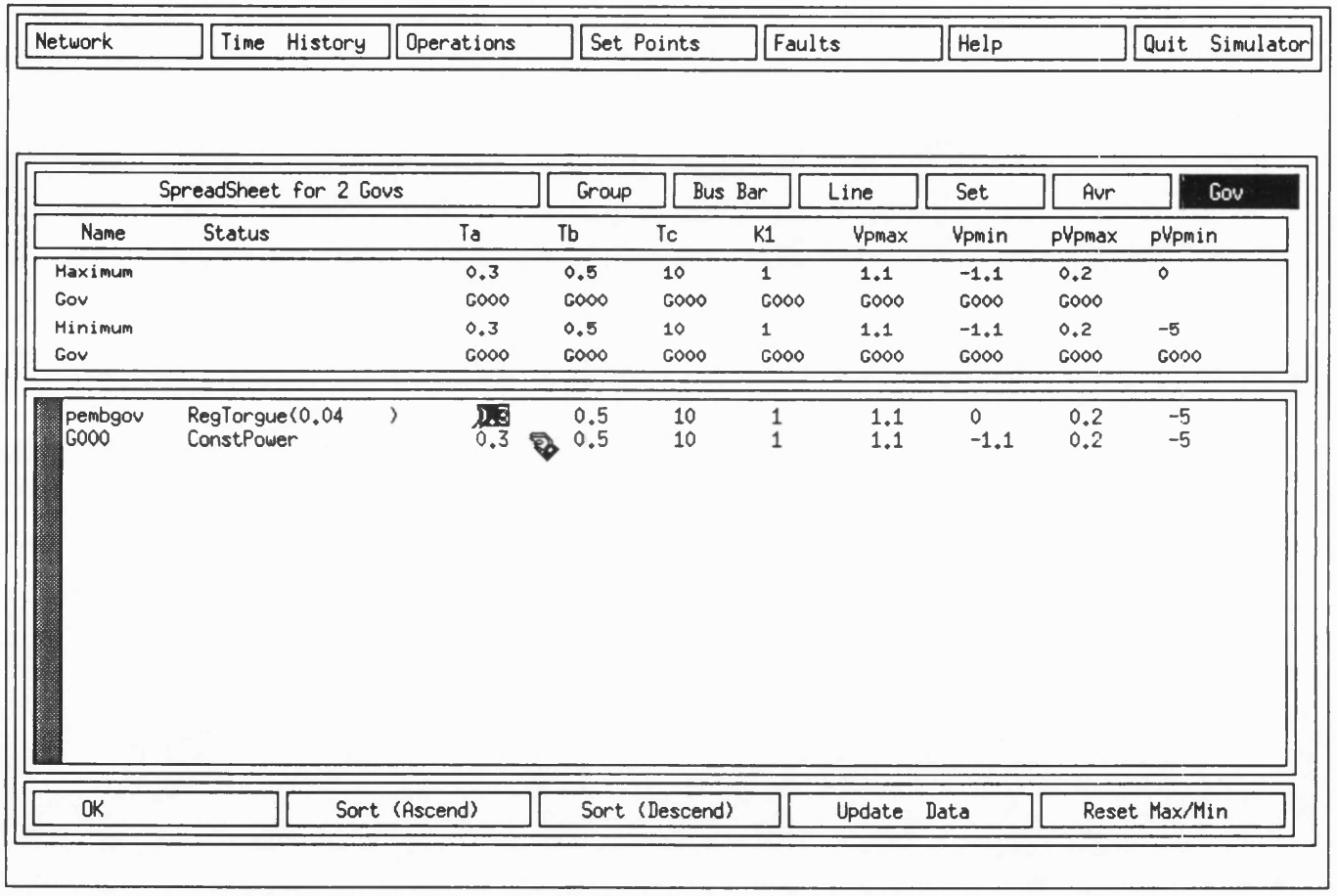


Figure 7.40: Spreadsheet of all the available governors

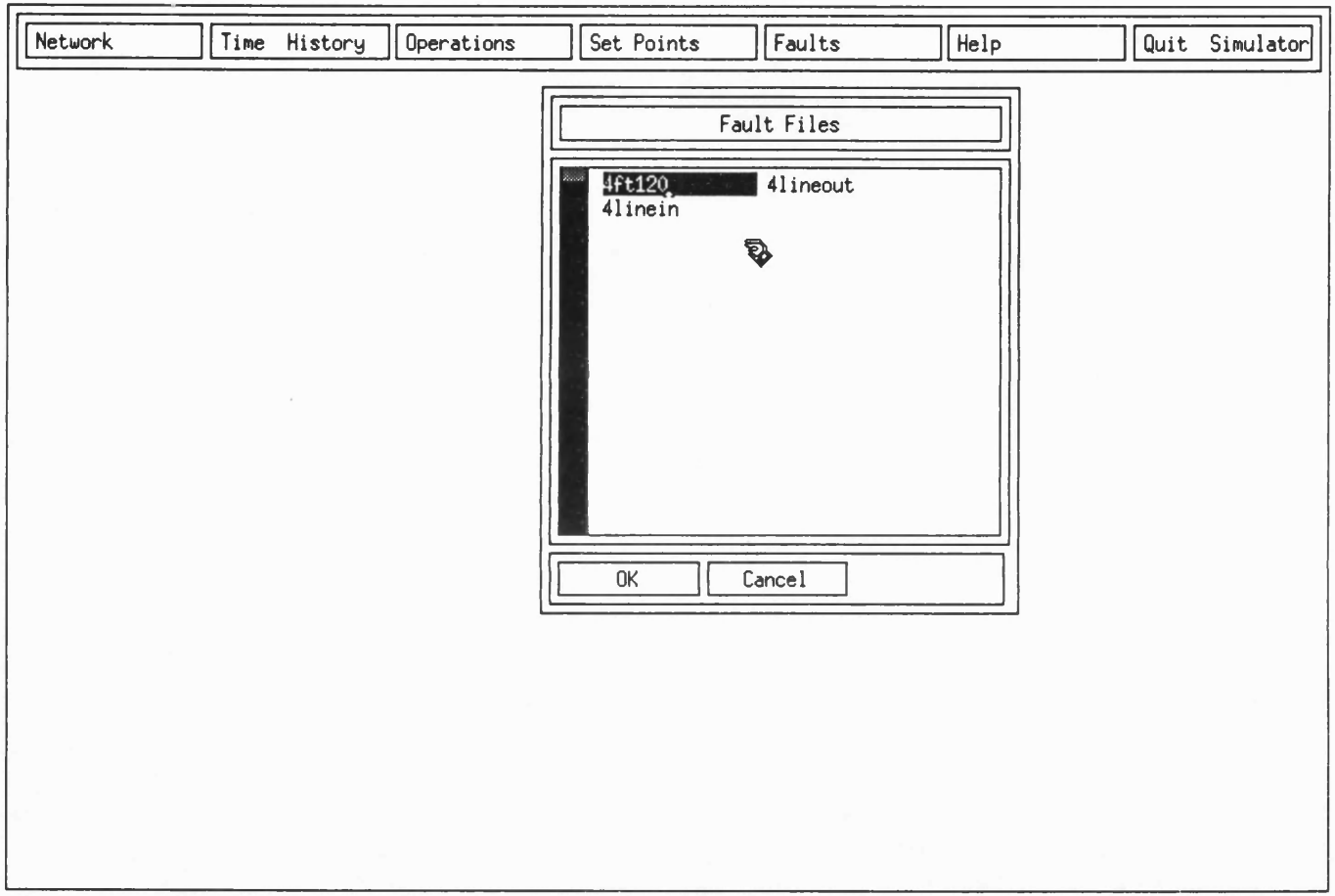


Figure 7.41: Menu of fault files

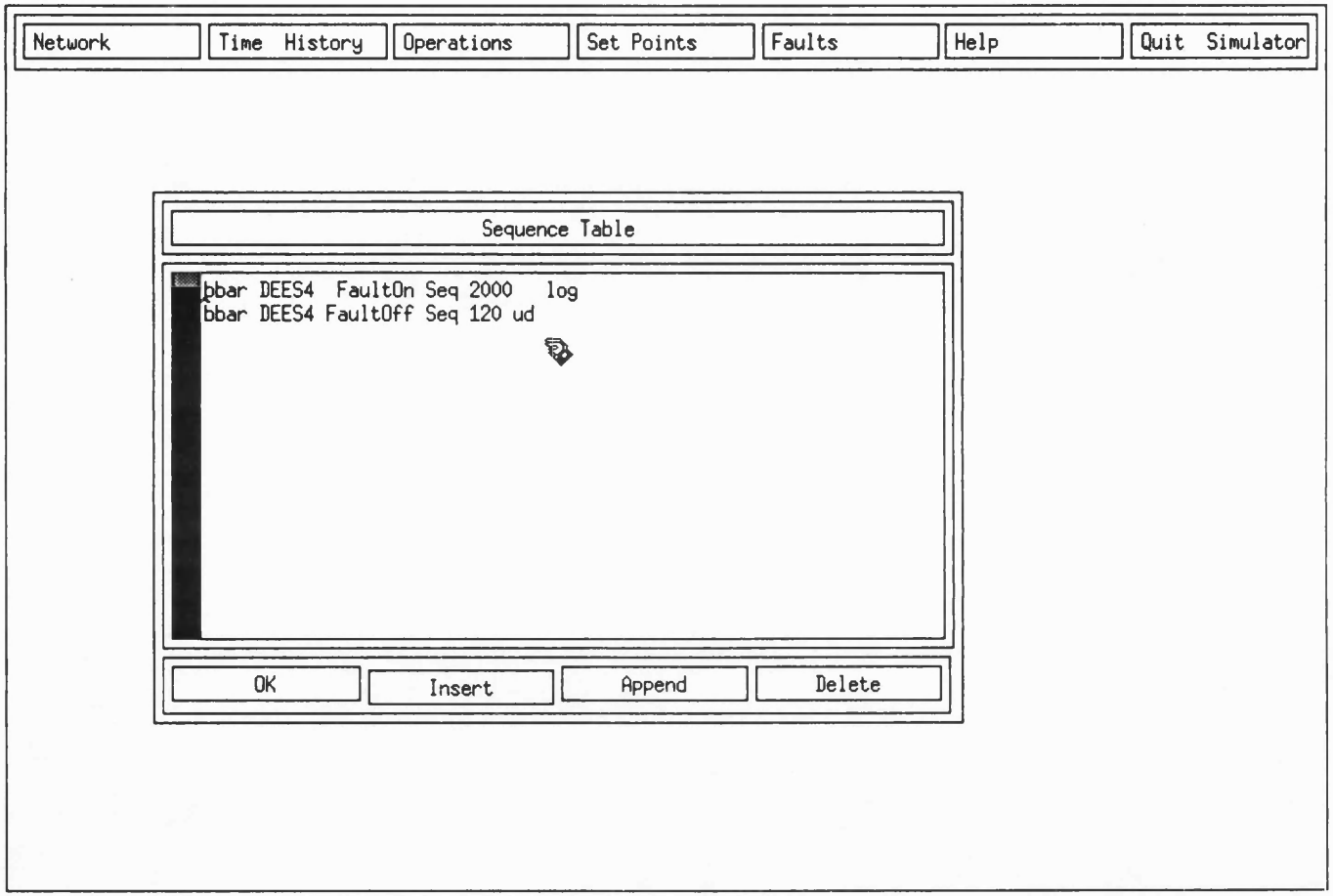


Figure 7.42: Table displaying the content of a selected fault file

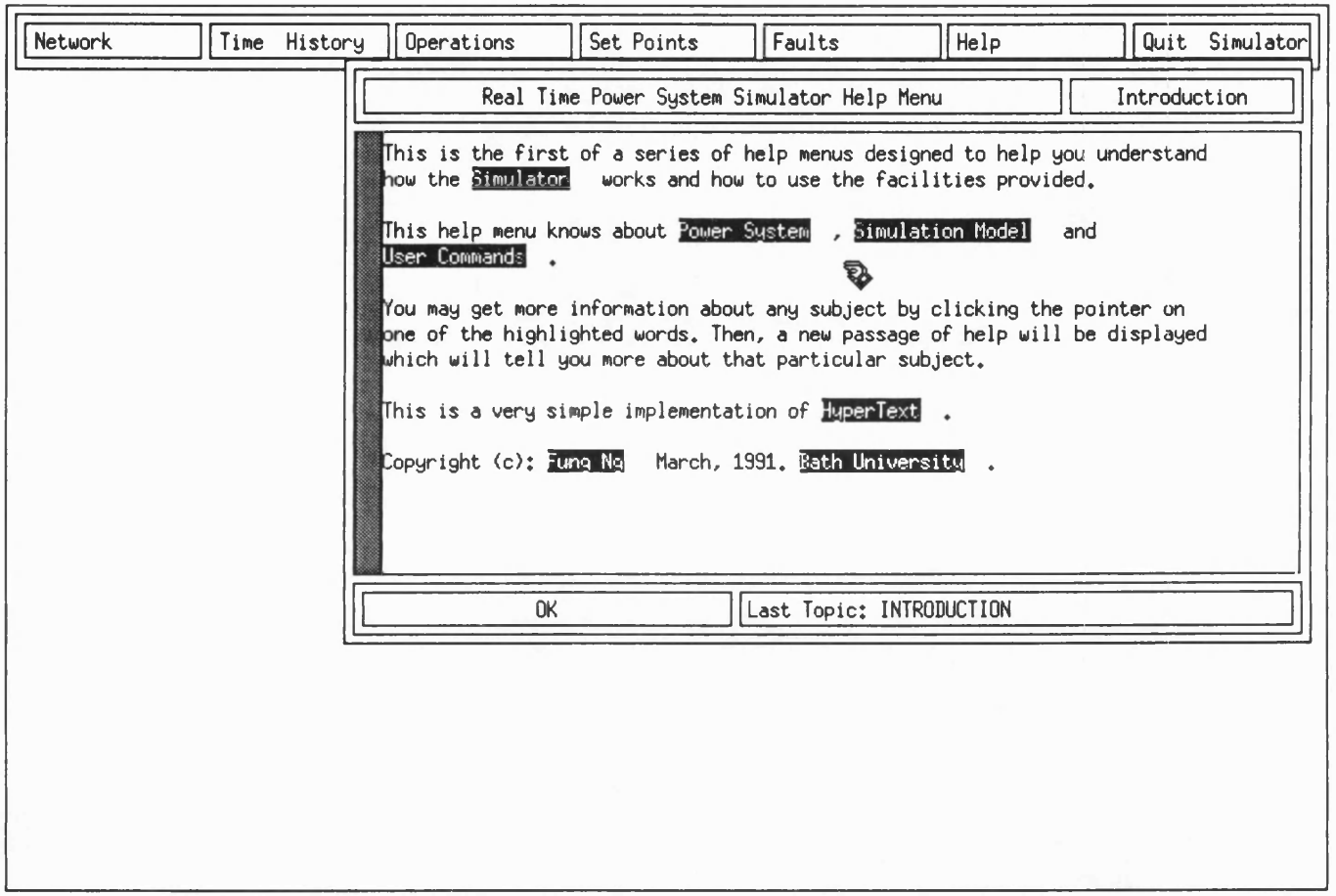


Figure 7.43: Hypertext help menu

Chapter 8

Further Work

The MMI is the first Graphical User Interface, GUI, developed for the present real time power system simulator. It opens up scope for further development in the field of man machine interface design for engineering problems.

Traditionally, the man machine interface has not played a major role in the development of power system simulators at Bath University [2, 3]. The present MMI brings to the user pleasant benefits of a carefully designed user interface. Unfortunately, some new problems, which have never been considered before, are created. For instance, mixing real time flicker free animation with a window system requires much work to solve the problem. The choice of menu items and how they should be positioned on the screen generates much debate. It is easier to refine a product than to create it from scratch. Once the user has more experience with the MMI, criticisms naturally arise. In order to serve the diversity of users, flexible methods can be constructed to make modifications of the interface easier. The introduction of the following features could be used to satisfy most user requirements:

1. It should be possible to design a window manager to take care of the user's

requirements during run time. The user can specify a number of interface features, e.g. colour of a window, from a pop up window during run time. This is better than customizing the interface by listing the user's requirements in a text file which is the common practice of most X Window System applications [65]. The proposed method is more dynamic and the user make changes easier. Sometimes, a user does not know what to change before the application is used. Obviously, it must also be possible to save the new modifications in a text file so that the user does not have to go through the whole customization process every time the application is used.

2. For more fundamental customization, such as the choice of menu items in a pop up menu and how they should be positioned, a more complex method must be adopted. The present MMI adopts the toolkit rather than the UIMS approach [1]. It should be feasible to develop a UIMS to function on top of the present system. Ambitious user should be allowed to make more basic alterations of the interface by interacting with the UIMS prior to the running of the simulator. In any case, a default interface should be provided to cater for those users who are only interested in running the simulator and have no need for changing the interface in any way other than some cosmetic changes which are accommodated by the window manager as described above. The present MMI is a very good example of how a user friendly and powerful interface can be built.

Lastly, it is possible to alter the "look and feel" of an X application by adopting a new widget set [2]. The present MMI was implemented with the Athena Widget Set [70] which offers the most basic functions expected from a widget set. Unfortunately, applications written with this widget set are not particular

[1] Toolkits and User Interface Management Systems (UIMS) are described in Chapter "User Interface Design" and Chapter "The Interface"

[2] Refer to Chapter "X Window System"

beautiful in appearance. In order to "beautify" the interface, the Motif Widget Set [105] is a very good choice. As well as being a powerful widget set offering more features than the Athena Widget set, it is fast gaining popularity among today's X applications. By adopting Motif, the new MMI should ease anxiety from the new users.

Chapter 9

Conclusion

A user friendly Man Machine Interface has been successfully implemented for a real time power system simulator running on the multiprocessor transputer system with high resolution bit-mapped colour graphics. The whole system is used to simulate and study the electro-mechanical transient behaviours of the British National Grid transmission system in real time.

The MMI is made up of two parts: the application interface and the user interface. The application interface is a library of application specific function calls which are responsible for interacting with the user interface in order to provide services that alter and obtain application data. The user interface is responsible for servicing the user requests in controlling the Simulator to carry out power system study. A set of carefully researched design guidelines have been adopted to ensure the usability of the interface. They are obtained from studying the human psychology, colour theory and various modern interface techniques. In addition, comprehensive facilities are provided to help the user in carrying out power system studies.

The MMI is implemented as a WYSIWYG type interface which has a close re-

lationship between user expectations and graphics output. In this way, the user can understand the MMI more and is in better control of it. Direct Manipulation has been shown to be a powerful technique for human-computer interaction as it successfully associate user actions and the application functions together. The user can readily interact with application objects by directly manipulating it using a mouse pointer. The application outputs are presented vividly in graphical forms which can be easily understood.

The X Window System has been adopted to implement the MMI. It offers a rich set of interface objects, e.g.pop-up menus and slide bars, as well as a comprehensive library of programming functions, Xlib, in aid to implement a user interface. X does not dictate how an interface should look like or behave. It only provides mechanisms. Hence, the "look and feel" of a user interface depends entirely on the programmer or on the choice of a Window Manager.

Unfortunately, X was not designed to provide animated graphics. A carefully designed method is used to implement flicker-free animation: a library of low level graphics routines are written to draw graphics directly on the video memory of the graphics processor, VSC. Two pages of memories are provided for double buffering, i.e. one page to display animation on the monitor while the other page was updated in the background. The result has been satisfactory in that the animation produced is fast and flicker- free. Careful compromise has been reached in bringing X and the animated graphics together.

The MMI has been implemented successfully in meeting the objective of providing a powerful and user friendly interface to a complex power system simulator. As it is the first project of its kind for the simulator developed at the School of Electrical Engineering, Bath University, it indicates that a proper user interface

is both feasible and desirable. As well as demonstrating how a user interface can be implemented, the MMI project also opens up scopes for further development by studying the strength and weakness of the present MMI. In the future, power system engineers should be benefited from a carefully constructed man machine interface mediating between the simulator and the users.

References

[1] T. Berry, K. Chan, R. W. Dunn, and F. Ng. Real time power system training simulator. In *25th UPEC Proceedings*, pages 671–674, 1990.

[2] L. A. Dale. *Real Time Modelling of Multi-machine Power System.* PhD thesis, Bath University, School of Electrical Engineering, 1986.

[3] T. Berry. *Real Time Modelling of Complex Power System using Parallel Processing.* PhD thesis, Bath University, School of Electrical Engineering, 1989.

[4] T. Hagen. A conceptual basis for graphical input and interaction. In *Methodology of Interaction*, pages 239–246. Amsterdam, 1980.

[5] Macintosh is a registered trademark of apple computers, inc.

[6] J. Whiteside. User performance with command menu, and iconic interface. In *The OX Association for Computing Machinery*, pages 185–191, Apr. 1985.

[7] The x window system is a registered trademark of the massachusetted institute of technology.

[8] Prentice-Hall. *The Helios Operating System*, 1989.

[9] Prentice-Hall. *Transputer Reference Manual*, 1988.

[10] Philips Components Ltd. *VSC: SCN66470 Video and System Controller*, 1988.

[11] G. A. Miller. The magic number 7, plus or minus 2: some limits on our capacity to process information. In *Psychological Review*, volume 63, pages 81–97, 1956.

[12] H. Thimbleby. What you see is what you have got - a user engineering principle for manipulative display. In *ACM Conference on Software, Ergonomics*, 1983.

[13] J. Rassmussen. Skills, rules and knowledge; signals, signs and symbols and other distinctions in human performance models. In *IEEE Trans Systems, Man and Cybernetics*, pages 257–266.

[14] B. Shneiderman. Direct manipulation: a step beyond programming languages. In *IEEE Computer*, pages 57–69, Aug. 1983.

[15] G. Polya. *Mathematical Discovery.* John Wiley, 1981.

[16] W. E. Hick. On the rate gain of information. In *Q. J. Experiemental Psychology*, volume 4, pages 11–26, 1952.

[17] K. R. Popper and J. C. Eccles. *The Self and its Brian.* Springer International, 1977.

[18] M. Green. A survey of 3 dialogue models. In *ACM Transaction on Graphics*, volume 5, pages 244–275, July 1986.

[19] B. Shneiderman. *Designing the User Interface: Strategies for Effective Human Computer Interactions.* Addison-Wesley, 1987.

[20] P. Heckel. *The Elements of Friendly Software Design.* Warner Books, 1984.

[21] E. L. Hutchins. Direct manipulation interfaces. In *User centered systems design: new perspectives on Human Computer Interactions.* Lawrence Erlbaum Associates, Hillsdale, N.J., 1986.

[22] G. Polya. *How to solve it.* Doubleday, 1956.

[23] M. Montessori. *The Montessori Method.* Schocken, 1964.

[24] J. Bruner. *Toward a Theory of Instruction.* Harvard University Press, 1966.

[25] J. M. Carroll, J. C. Thomas, and A. Malhotra. Presentation and representation in design proble solving. In *British Journal of psychology*, volume 71, pages 143–153, 1980.

[26] S. Papert. *Mindstorms: Children, Computers and Powerful Ideas.* Basic Books Inc., 1980.

[27] R. Arnheim. *Visual Thinking.* University of California Press, 1972.

[28] M. Wertheimer. *Productive Thinking.* Harper and Row, 1959.

[29] J. Ossner. Transnational symbols: The rule of pictograms and models in the learning process. In *In Designing User Inerface for International use*, pages 11–38. Elsevier Science Publication, Amsterdam, 1990.

[30] P. Sukaviriya and L. Morgan. User interface for asia. In *In Designing User Inerface for International use*, pages 11–38, Amsterdam, 1990. Elsevier Science Publication.

[31] G. M. Murch. Human factors of colour displays tutorial no. 24 notes. In *Siggraph*, July 1987.

[32] L. W. MacDonald. Using colour effectively in displays for computer-human interface. In *Displays*, July 1990.

[33] G. M. Murch. Physiological principles for the effective use of color. In *IEEE Computer Graphics and Application*, pages 49–53, Nov. 1984.

[34] J. A. Wise and N. C. Abi-Samra. *The Design of Display Systems for Electrical Control Centres.* Westinghouse Electric Corporation.

[35] D. G. Bobrow, S. Mittal, and M. J. Stefik. Expert systems: Perils and promise. In *The OX Association for Computing Machinery*, pages 880–894, Sept. 1986.

[36] J. A. Sutton and R. H. Sprague. A study of display generation and management in interactive business applications. Technical Report 31804, IBM San Remo Lab, 1978.

[37] W. D. Hurley and J. L. Sibert. Modelling user interface–application interfaces. In *IEEE Software*, pages 71–77, Jan. 1989.

[38] H. Lieberman. There is more to menu systems than meets the screen. In *Computer Graphics*, volume 19, pages 181–189, 1985.

[39] H. W. Dzida and W. D. Itzfeldt. Factors of user perceived quality of interactive systems. Technical Report 40, Institut Fur Software Technolgie, 1978.

[40] J. Rhyne. Tools and methodology for user interface development. In *Computer Graphics*, pages 78–87, Apr. 1982.

[41] E. Lee. User-interface development tools. In *IEEE Software*, pages 31–35, May 1990.

[42] B. A. Myers. User-interface tools: Introduction and survey. pages 15–23, Jan. 1989.

[43] D. R. Olsen. A context for user interface management. In *IEEE Computer Graphics and Application*, pages 33–42, Dec. 1984.

[44] R. L. Read and M. L. Smith. A light-weight uims. In *Software-Practice and Experience*, volume 21, pages 13–33, Jan. 1991.

[45] D. J. Kasik, M. A. Lund, and H. W. Ramsey. Reflection on using a uims for complex applications. In *IEEE Software*, pages 54–89, Jan. 1989.

[46] R. Hartson. User-interface management control and communication. In *IEEE Software*, pages 62–70, Jan. 1989.

[47] J. D. Foley. Models and tools for the designers of user-computer interfaces. Technical report, Dept. EE and CS, The George Washington University, Washington, DC, USA.

[48] P. J. Hayes and P. A. Szekely. Design alternatives for user interface management system based on experience with cousin. Technical report, Computer Science Dept., carnegie-Mellon University, Pittsburgh, PA, USA.

[49] J. McCormack and P. Asente. X 11 toolkit for the x window manager. In *Proc ACM SIGGraph Symp., User-Interface Software*, pages 46–55, 1988.

[50] P. S. Barth. Grow: An object–oriented approach to graphical interface. In *ACM Transaction on Graphics*, pages 142–172, Apr. 1986.

[51] P. Szekely and B. Myers. Coral: A user–interface toolkit based on graphical objects and constraints. pages 36–45, Nov. 1988.

[52] M. A. Linton, J. M. Vlissides, and P. R. Calder. Composing user interfaces with interviews. In *IEEE Computer*, Feb. 1989.

[53] Windows is a registered trademark of microsoft.

[54] Open windows is a registered trademark of sun microsystems.

[55] Presentation manager is a registered trademark of microsoft.

[56] Dec windows is a registered trademark of digital equipment corp.

[57] Gem is a registered trademark of digital research.

[58] B. Liskov and R. Scheifler. Guardian and action: Linguistic support for robust, distributed programs. In *ACM Trans. Program Lang. Syst*, volume 5, pages 381–404, July 1983.

[59] E. Balkovich, S. Lerman, and R. P. Parmelee. Computing in higher education: The athena experience. In *The OX Association for Computing Machinery*, volume 28, pages 1214–1224, Nov. 1985.

[60] P. Asente. *W: reference manual.* Dept Computer Science, Stanford University, Calif, U.S.A., 1984.

[61] K. A. Lantz and W. Norwicki. Structured graphics for distributed systems. In *ACM Trans. Graph*, volume 3, pages 23–51, Jan. 1984.

[62] D. Cheriton. The v kernel: A software base for distributed systems. In *IEEE Software*, volume 1, pages 19–42, Apr. 1984.

[63] Unix is a registered trademark of at&t.

[64] Prentice-Hall. *Helios X Window Manual*, 1990.

[65] R. W. Scheifler and J. Gettys. The x window system. In *ACM Transaction on Graphics*, volume 5, pages 79–109, Apr. 1986.

[66] R. W. Scheifler. *X Window System Protocol*, 1987.

[67] J. Gettys, R. Newman, and R. W. Schiefler. *Xlib - C Language Interface.* Massachusetted Institute of Technology, 1987.

[68] J. McCormack, P. Asente, and R. Swick. *X Toolkit library - C Language Interface.* Massachusetted Institute of Technology, 1987.

[69] D. S. H. Rosenthal. A simple x11 client program, or, how hard can it really be to write 'hello, world'. In *USENIX Conference Proceedings*, 1987.

[70] R. R. Swick and M. S. Ackerman. The x toolkit: more bricks for building user-interfaces, or, widgets for hire. In *USENIX Conference Proceedings*, 1987.

[71] T. Berry. Real time simulation of power system transient behaviours. In *IEE 3rd International Conference on Power System Monitoring and Control Conference*, 1991.

[72] R. W. Dunn. A new architecture of high performance parallel computer for use in condition monitoring of large diesel engine. In *IEE Conference Publication*, Sept. 1989.

[73] MetaComco. *Introduction to Tripos*, 1986.

[74] M. B. Daley. *A Link Topology Controller for a Sixteen Transputer Multiprocessor System.* BSc thesis, Bath University, School of Electrical Engineering, 1988.

[75] Ims c004 programmable link switch, 1987.

[76] M. Hafeez. *An Expandable Input/Output and Graphics System for Distributed Memory Parallel Computer.* PhD thesis, Bath University, School of Electrical Engineering, 1990.

[77] Philips Components Ltd. *16/32-Bit Highly-integrated Microprocessor SCC68070 User Manual Part1-Hardware*, 1988.

[78] IXI. *MIT X Window System Manual Set: 0, 1, 2, 3*, 1988.

[79] Philips Components Ltd. *Microcore: An evaluation board for 68070 and VSC*, 1987.

[80] G178 colour palette, data sheet, 1987.

[81] Logitech. *Logitech mouse user's manual*, 1988.

[82] IBM AT is a registered trademark of the international business machines.

[83] D. May. Occam. In *SIGPLAN notices*, volume 18, pages 69–79, Apr. 1983.

[84] S. L. Harbison and G. L. Steele. *C: A Reference Manual.* Prentice-Hall, 1987.

[85] Sun workstation is a registered trademark of sun microsystem, inc.

[86] J. Powell and N. Garnett. Helios performance measurement. Technical Report 22, Perihelion Software, 1990.

[87] J. M. Undril. Interactive computation in power system analysis. In *IEEE Proceedings*, volume 2, pages 1009–1018, July 1974.

[88] U. G. Knight. Computers in power system planning. In *IEEE Proceedings*, volume 62, pages 872–883, July 1974.

[89] F. L. Alvarado, S. K. Mong, and M. K. Enns. A fault program with macros, monitors, and direct compensation. In *IEEE Trans. on Power Apparatus and Systems*, volume 104, pages 1109–1120, May 1985.

[90] R. Fujiwara and Y. Kohno. User-friendly workstation for power systems analysis. In *IEEE Trans. on Power Apparatus and Systems*, volume 104, pages 1370–1376, June 1985.

[91] C. H. Lo, M. D. Anderson, and E. F. Richards. An interactive power system analyser with graphics for educational use. In *IEEE Transactions on Power Systems*, volume 2, pages 174–181, May 1986.

[92] H. Diab, M. Yehia, and I. Abou-Hassan. Parallel computer graphic simulation of the lebanese electric power system. In *IEEE Computer Applications in Power*, pages 38–42, 1989.

[93] D. C. Yu, S. Chen, and R. F. Bischke. A pc oriented and graphical simulation package for power system study. In *IEEE Transactions on Power Systems*, volume 4, pages 353–360, Feb. 1989.

[94] R. Bonert. Interactive simulation of dynamic systems on a personal computer to support teaching. In *IEEE Transactions on Power Systems*, volume 4, pages 380–383, Feb. 1989.

[95] M. Daneshdoost and R. Shaat. A pc based integrated software for power system education. In *IEEE Transactions on Power Systems*, volume 4, pages 1285–1292, Aug. 1989.

[96] G. Zhang and A. Bose. Scenario building for operator training simulators using a transient stability program. In *IEEE Transactions on Power Systems*, volume 4, pages 1542–1549, Oct. 1989.

[97] S. Chan. Interactive graphics interface for power system network analysis. In *IEEE Computer Applications in Power*, pages 34–38, 1990.

[98] S. S. Miller and M. R. Ward. Implementation and use of a transient stability post-processor. In *IEEE Computer Applications in Power*, pages 19–24, 1990.

[99] D. C. Yu, S. Chen, and R. J. Kalscheur. A pc based interactive graphical simulation and analysis package for a power plant electrical auxiliary system. In *IEEE Transactions on Power Systems*, volume 5, pages 628–634, May 1990.

[100] W. Long. Emtp: A powerful tool for analyzing power system transients. In *IEEE Computer Applications in Power*, pages 36–41, 1990.

[101] K. F. Chan and J. Ding. Interactive network planning and analysis on a personal computer. In *IEEE Computer Applications in Power*, pages 43–47, Jan. 1990.

[102] B. Valiquette. Microcomputer based power network control centre simualtor for education. In *IEEE Transactions on Power Systems*, volume 5, pages 474–481, May 1990.

[103] M. Ahmadiaan and A. Brameller. Power system operator training simulator. In *Proceedings of the 10th Power System Computation Conference*, pages 717–724, Aug. 1990.

[104] L. Koved and B. Shneiderman. Embedded menus: Selecting items in context. In *The OX Association for Computing Machinery*, volume 29, pages 312–318, Apr. 1986.

[105] R. J. Rost. *X and Motif Quick Reference Guide.* DEC Press, 1990.

[106] F. Ng. *Computer Aided Design in Large Power System Simulation.* BSc thesis, Bath University, School of Electrical Engineering, 1988.

[107] E. Marr. Eace02 - electromechanical equivalents for power system stability studies. Technical Report CC/R252, CEGB Computing Services Department.

[108] F. J. Gilchrist. Race01–a.c. reduction program. Technical Report CC/P510, CEGB Computing Services Department.

[109] E. Roe. Prasm–linear analysis of transient stability program. Technical Report CC/N784, CEGB Computing Services Department.

[110] H. W. Dommel and N. Sato. Fast transient stability solutions. In *IEEE Trans. on Power Apparatus and Systems*, pages 1643–1650, 1972.

[111] B. Stott. Power system dynamic response calculations. In *IEEE Proceedings*, volume 67, pages 219–241, 1979.

[112] J. Arrillaga, C. P. Arnold, and B. J. Harker. *Computer modelling of electrical power systems.* John Wiley, 1983.

[113] Computer representations of exciatation systems. In *IEEE Trans. on Power Apparatus and Systems*, volume 87, pages 1460–1464, 1968.

[114] Dynamic models for steam and hydro turbines in power system studies. In *IEEE Trans. on Power Apparatus and Systems*, volume 1973, pages 1904–1915, 1973.

[115] Y. B. Lee. *Sensitivity and optimal control studies of power systems.* PhD thesis, Bath University, School of Electrical Engineering, 1975.

[116] H. Lu. *Optimization studies of single machine power system.* PhD thesis, Bath University, School of Electrical Engineering, 1979.

[117] P. A. Hazell. *Co-ordinate excitation control and governing of turbogenerators.* PhD thesis, Bath University, School of Electrical Engineering, 1981.

[118] F. Busemann and W. Casson. Results of full-scale stability tests on the british 132kv grid system. In *IEE Proceedings*, volume 105, pages 347–362, 1958.

[119] E. C. Scott. Multi-generator transient stability performance under fault conditions. In *IEE Proceedings*, volume 110, pages 1051–1064, 1963.

[120] G. Shackshaft and R. Neilson. Results on stability tests of underexcited 120mw generator. In *IEE Proceedings*, volume 119, pages 175–18, 1972.

[121] D. W. Olive. New techniques for the calculation of dynamic stability. In *IEEE Trans. on Power Apparatus and Systems*, volume 85, pages 767–777, 1966.

Appendix A

Starting up the Simulator

The content of the shell script file, m4start, as used in Chapter 7, is given below:

#helios/bin/shell
remote -d 02 pownet
remote -d 02 read study/m4b6
remote -d 01 mmi -geometry 718x478+0+0
-helpfile intro -network pic/fourmach

The functions of the file are:

1. loading the power system network server onto processor 02 and initializing the Simulator.
2. loading the read server onto processor 02 and reading in the input study file: study/m4b6.
3. creating a number of machine tasks from the input study file.
4. distributing the machine tasks around the network of transputer processors.

5. loading the Interface onto processor 01.

6. initializing X, screen layout and the Interface with the help file and network diagram.

The user can customize the shell script file for different power system study. The choices of screen layout, help file and initial network diagram are also controllable by the user.

Appendix B

Power System One-line diagram

Introduction

The Network option in the MMI needs to display a one-line diagram of the power system under study. The diagram must be prepared by the user before the MMI is started. The diagram can be constructed either by using an interactive picture editor [106] or by editing a text file specifying the picture. This appendix illustrates how a one-line diagram can be specified by the latter method.

The whole one-line diagram is confined to a square which has a dimension of 1500x1500 points. The origin of the square is located at the bottom left hand corner. A smaller moving square of dimension 500x500 points is used to specify which portion of the whole diagram is to be displayed. When the MMI is brought up, the default is to move the smaller square to the bottom left hand corner of the larger square, thus covering the area from (0,0) to (500,500). Anything outside this boundary is clipped.

Commands

The followings show the valid keywords and their corresponding arguments in specifying a one-line diagram:

Get arg1

A large one-line diagram can be made up of a number of smaller sub-pictures which are contained in separate files. A master file is used to specify all the smaller files in making up the whole diagram. This keyword specifies that the MMI should read in a new file, whose name is given by arg1, and uses the content of the file as the new input. When this new file is finished, the MMI should carry on reading in the interrupted file until the end. The whole process can be iterative.

Picture x arg1 y arg2

If the "Picture" keyword is not present, the input graphics coordinates are assumed to be relative to the origin, i.e. (0,0). When a large network diagram is to be specified, it is usually made up of a few smaller files which might be constructed one by one with the origin located at (0,0). The "picture" keyword informs the MMI that the incoming graphics has a different origin and it is located at (arg1,arg2).

Item arg1

The data structure of the one-line diagram is made up of a list of items. Each item consists of a number of elements which might be a line, a box, a circle or a text string. The keyword "item", whose name is given by arg1, indicates that all

the elements following this declaration belong to this item.

Line arg1 arg2 arg3 arg4 arg5

Specify a line element starting from (arg1,arg2) to (arg3,arg4) with the colour given by arg5.

Box arg1 arg2 arg3 arg4 arg5

Specify a box element starting from (arg1,arg2) to (arg3,arg4) with the colour given by arg5.

Circle arg1 arg2 arg3 arg4

Specify a circle element with centre (arg1,arg2) and a radius of arg3 with the colour given by arg4.

Text arg1 arg2 arg3 arg4

Specify a text at (arg1,arg2) whose content is specified by arg3, with the colour given by arg4.

End

Specifies the end of the current input file.

Example

The following is the text file used to specify the one-line diagram of the four machine power system study [1]. The circles represent the machine groups, the rectangle represent the busbars and the lines represent the transmission lines. The pointer within a circle is used to indicate the rotor angle of the machine group with respect to its busbar voltage angle. The magnitude of a busbar voltage is displayed inside a rectangle. The two numbers beside a circle represent the real and reactive power of a machine group. A positive value indicates that the machine group is generating power and a negative number indicates that power is being absorbed.

```
Item "SCOT4"
box     420    400    480    460    blue
text    430    450    "SCOT4"       white

Item "DEES4"
box     280    260    340    320    blue
text    290    310    "DEES4"       white

Item "PENT4"
box     120    260    180    320    blue
text    130    310    "PENT4"       white

Item "DINO4"
box     110    100    170    160    blue
text    120    150    "DINO4"       white

Item "CEGB4"
box     420    100    480    160    blue
text    430    150    "CEGB4"       white

Item "NWAL4"
box     280    100    340    160    blue
text    290    150    "NWAL4"       white
```

[1]The diagram is used in Chapter "Features of the MMI"

```
Item "DINORWIG"
circle  142    50    30              cyan
line    140    82    140    100      cyan
text    122    12    "DINORWIG"      white

Item "NWALES"
circle  310    50    30              cyan
line    310    82    310    100      cyan
text    290    12    "NWALES"        white

Item "CEGB"
circle  450    50    30              cyan
line    450    82    450    100      cyan
text    450    12    "CEGB"          white

Item "SCOTLAND"
circle  340    430    30              cyan
line    360    430    420    430      cyan
text    320    382    "SCOTLAND"      white

Item "DEES4-PENT4:L1"
line    280    300    180    300      cyan

Item "DEES4-PENT4:L2"
line    280    290    180    290      cyan

Item "PENT4-:"
line    120    290    100    290      cyan
line    110    220     90    220      cyan
line    100    240    100    220      cyan
box      95    240    105    270      Magenta
line    100    270    100    290      cyan

Item "DINO4-PENT4:L1"
line    140    160    140    260      cyan

Item "DINO4-PENT4:L2"
line    130    160    130    260      cyan

Item "DEES4-PENT4:L3"
line    280    270    180    270      cyan

Item "DEES4-CEGB4:"
line    340    290    440    290      cyan
line    440    290    440    160      cyan
```

```
Item "DEES4-NWAL4:"
line    310    260    310    160     cyan

Item "DEES4-SCOT4:"
line    340    300    440    300     cyan
line    440    300    440    400     cyan

Item "PENT4-CEGB4:"
line    170    215    170    260     cyan
line    170    215    430    215     cyan
line    430    160    430    215     cyan

Item "PENT4-NWAL4:"
line    160    260    160    200     cyan
line    300    200    160    200     cyan
line    300    200    300    160     cyan

Item "PENT4-SCOT4:"
line    150    340    150    320     cyan
line    150    340    430    340     cyan
line    430    340    430    400     cyan

Item "CEGB4-NWAL4:"
line    420    120    340    120     cyan

Item "CEGB4-SCOT4:"
line    470    160    470    400     cyan

Item "NWAL4-SCOT4:"
line    330    200    330    160     cyan
line    460    200    460    400     cyan
line    460    200    330    200     cyan

Item "title"
text    80    465    "Four Machine Power System"     White
line    75    450    300    450    red

end
```

Appendix C

Specifying a sequence for the power system

The Simulator provides commands to build up a complex sequence of modifications to the power system under study. The user can write a text file to contain all the commands [2]. Alternatively, the MMI provides a visual means to allow the user construct a sequence by interacting with the objects on the one-line diagram in the Network option. The former method is straight forward. However, the user must have a fair understanding of the whole power system network beforehand. The latter method allows the user to examine the geographical locations and connectivities of all the objects on the network. Besides, additional information about each object can be monitored as the simulation is running. This gives the user a dynamic control over the whole network. This appendix describes the procedure used to construct a sequence and all the commands available.

Constructing a sequence

Initiating a sequence

The user selects the "Start Sequence" on the Network Sub-menu [1] to initiate a sequence. The shape of the mouse pointer will be changed into a hand to confirm the user's action. The user can then specify a sequence by modifying the various power system objects.

Specifying a modification

The user uses the same method in modifying an object during a sequence as described in Chapter: "Feature of the MMI". After the modification, a new popup window is displayed. There are three further commands which can be selected. The user can click on three command buttons to toggle on and off the commands. A window is used to display the information about the object and the modifications made to it. The three commands are:

1. Ud: Although a modification has been made to the object, the Simulator is not informed of the change yet. It is possible to specify the modifications of a group of objects without updating the Simulator. The actual updating of all the simulation data is delayed until the "ud" command is issued.

2. Log: All simulation data is stored in log buffers. This commands forces the Simulator to flush the contents of the buffers so that new simulation data is stored.

[1] Refer to Chapter: "Feature of the MMI" for a description of the Network Sub-menu.

3. Seq: This command is used to specify the duration of a modification. For instance, it is possible to specify how long a busbar fault should last.

Ending a sequence

The user can end a sequence by selecting the "End Sequence" on the Network Sub-menu. The shape of the mouse pointer will be changed back into a large "X" to confirm the user's action.

Modifying an object

Modifying a Machine Group

The parameters of a machine group that can be modified are:

1. AVR reference.
2. Governor reference.
3. Tap ratio.
4. Real generating power.

Modifying a Busbar

The parameters of a busbar that can be modified are:

1. Real load power.
2. Reactive load power.
3. Busbar fault on.
4. Busbar fault off.

Modifying a Line

A line can be either:

1. Switched in.
2. Switched out.

An Example

The following is an example used to apply a 120ms busbar fault at the busbar DEES4, for the four machine study [2]. Comment lines are started with a "*".

```
* Flush the log buffer and wait for 2000ms.
* Then, apply a busbar fault at DEES4.
* Update the simulator data.

busbar DEES4 faulton seq 2000 log ud

* Wait for the busbar fault for 120ms.
* Then take off the busbar fault.

busbar DEES4 faultoff seq 120
```

[2] Refer to Appendix: "The Four Machine Power System".

Appendix D

Designing a Hypertext Help Menu

A hypertext help menu is made up of a number of help files each of which is identified by a keyword. A help file might contain a number of keywords which lead to further explanations of them. This helps to reduce the burden on the user's short term memory by leaving out temporarily irrelevant materials.

The MMI help system is made up of a number of files which are located under the directory HELPDIR. A text file, HELPDBASE, is used to map a keyword to its associated input file. When the MMI is started up, it searches for a default file, HELPDEFAULTFILE, and uses it as its first help file. These three macros, HELPDIR, HELPDBASE, and HELPDEFAULTFILE are specified in the header file, dir.h. Hence, by recompiling the corresponding files, new values can be assigned.

Every help file must be preprocessed by a program, prehelp, before it can be used. After running through the preprocessor, a new help file is generated for each original text file. The new file has a ".hlp" suffix. This process helps to

guarantee the validity of the help files before the MMI is started.

Inside a help file, each keyword is enclosed by a pair of square brackets, "[ ]". If a picture is to be included in the file, the picture file name is enclosed by a pair of curly brackets, "{ }". The format of the picture file is specified in Appendix: "Power System One-Line Diagram".

The following is an example of such a file:

```
A real time [Power System] simulator needs a user friendly
[Man Machine Interface]. The following is a picture of the
four machine power system:
{fourmach}
```

In this case, there are two keywords: "Power System" and "Man Machine Interface". The user can get more information about them by selecting the keywords. In addition, a picture is included in the menu whose name is specified by "fourmach".

The file, HELPDBASE, then specifies the mapping between a keyword and its corresponding input file. The following is an example:

```
"Power System"            power
"Man Machine Interface"   mmi
```

The keyword "Power System" leads to another help file, "power". Similarly, the keyword "Man Machine Interface" is associated with the file, "mmi".

When the MMI brings up the help menu, all the keywords within the text are highlighted. The user can use the mouse pointer to click on a keyword and bring

up a new help menu with a set of new keywords. The name of the most recently displayed help file is indicated in the bottom right hand corner of the help menu. This is used to provide a short cut for the user to refer back to the last shown page.

Appendix E

The Four Machine Power System

Introduction

The four machine power system was simplified from a large-scale CEGB system which was a representation for summer night light load conditions with a Scotland to CEGB transfer of five hundred mega watt [2]. The three lumped machines and the reduced network were produced by the CEGB using dynamic and static reduction methods [107,108]. An eigenvalue study [109] was applied to check that the reduced system retained the main system modes with sufficient accuracy. The reduced system was originally used to study the behaviours of the Dinorwig pump storage station under different operating conditions, following a dual circuit line outage between Deeside and Pentir. It was found that the system exhibited the main local, group and system modes of oscillation for the Dinorwig section [2]. Fig. 7.2 shows the schematic of the four machine system.

This appendix examines the methods used to model a power system and the

numerical techniques used to solve the system equations. This is a summary of the work of Dale [2].

Power System Modeling

A set of first order nonlinear differential equations are used to represent each machine and a set of algebraic equations are used to represent the interconnecting network. The Simulator represents each machine group by a second order AVR, third order governor and a fifth order voltage behind reactance model. Only balanced three phase conditions are modeled. A general purpose equivalent π circuit is used to represent a circuit branch so that either lumped parameter transmission lines or power transformers can be modeled. Norton equivalent sources, which include the action of a variable tap transformer, are used to represent the machine group for interfacing them to the network. Hence, the machine stator and transmission network are represented by a system of purely algebraic equations which couple the machine differential equations. The time response calculation is then a differential-algebraic initial value problem. The general form of such a problem is then:

$$py = \mathcal{F}(y, x) \tag{E.1}$$

$$0 = \mathcal{G}(y, x) \tag{E.2}$$

where y is a vector of integrable variables such as machine and control system states, x is a vector of the non-integrable algebraic variables and p represents the derivative with respect to time. In general $\mathcal{F}$ and $\mathcal{G}$ are non-linear functions and

thus the non-integrable network variables cannot be eliminated algebraically.

Synchronous Machine Model

The model is based on an idealized machine which is symmetrical about the axes of the field windings and that of the interpolar space, known as the direct and quadrature axes respectively. The original three phase quantities along these axes are resolved by Park's transformation.

As far as the idealization of the machine is concerned, hysteresis, saturation and eddy currents are neglected. Besides, a sinusoidal stator flux distribution around the air gap is assumed for the mutual effects with the rotors. For transient stability, only one shunt circuited winding in each axes is used to represent damper windings and lumped eddy current effects.

Subsequently, a seventh order machine is obtained and it is made up of a set of equations in terms of voltage behind subtransient reactance [2]. This model is particularly useful in simulation as the machine equivalent voltages are states and it is not necessary to compute them for the network calculations. To simplify further, the seventh order model is reduced to fifth order [2].

Machine Torque Equation

$$T_e = E''_d \cdot I_d + E''_q \cdot I_q - \left(X''_d - X''_q\right) \cdot I_d \cdot I_q \tag{E.3}$$

Swing Equations

$$p\omega = \left(T_m - T_e - T_{lo}\right)/M \tag{E.4}$$

$$p\delta = \omega \tag{E.5}$$

Electrical equations

$$pE''_d = \left((X_q - X''_q) \cdot I_q - E''_d\right) / T''_{qo} \quad \text{(E.6)}$$

$$pE'_q = \left(V_f - (X_d - X'_d) \cdot I_d - E'_q\right) / T'_{do} \quad \text{(E.7)}$$

$$pE''_q = \left(E'_q - (X'_d - X''_d) \cdot I_d - E''_q\right) / T''_{do} \quad \text{(E.8)}$$

$$V_d = E''_d + X''_q \cdot I_q - R_a \cdot I_d \quad \text{(E.9)}$$

$$V_q = E''_q - X''_d \cdot I_d - R_a \cdot I_q \quad \text{(E.10)}$$

The Network Calculation

As the transmission network dynamic effects are neglected, a steady-state nodal analysis of the network can be applied to E.2 and yield the following specific form:

$$I(E, V) = \mathbf{Y} \cdot V \quad \text{(E.11)}$$

where I is a vector of current injections. I is determined from the machine internal voltages E and/or busbar voltages V in the case of voltage dependent loads. $\mathbf{Y}$ is the network nodal admittance matrix which is sparse for most large power systems. The elements of this matrix are fixed with respect to time, under a given switching condition.

Each branch of the transmission network may be represented by the general purpose single phase equivalent π circuit if balance three phase operations and symmetrical three phase faults are assumed.

To interface the synchronous machine model, the stator equations are referred

into the network reference frame and forms the Norton equivalent source. The resulting equations form a complicated admittance matrix which requires modification every time step and the network matrix cannot be represented by complex numbers but a 2x2 array for each admittance term is needed.

To reduce the computing effort required to solve the network equation, a constant complex matrix is formed. The saliency effect is included by adjusting the machine current injection iteratively. Unfortunately, convergence of this method can be troublesome [110–112] and it may be necessary to add extra current injection to give reliable results. To eliminate the need for iterative network solution completely, subtransient saliency in the fifth order model is neglected. In this way, the nodal admittance matrix is then complex, sparse and constant for a given network condition. Faults, line outages and transformer tap changes must be obtained by recalculating $\mathbf{Y}$ in E.11.

Control Systems

The synchronous machine excitation system and prime mover characteristics must be included to complete the basic power system model. The general practice is to represent these systems using a number of standard linearized models [113, 114]. The Simulator uses simplified AVR and governor models similar to those used in the previous optimization studies [115–117].

Neglecting rate and position limits, the differential equations for these control systems are linear in their state variables and control input quantities and can be arranged as:

$$pV_f = -\frac{K_g}{T_g}V_s - \frac{1}{T_g}V_f + \frac{K_g}{T_g}V_{err} \quad \text{(E.12)}$$

$$pV_s = \frac{-1}{T_s}V_s + \frac{K_s}{T_s}pV_f \quad \text{(E.13)}$$

$$V_{err} = V_{ref} - V_t \quad \text{(E.14)}$$

Equations E.12 and E.13 can be combined to give the first order equation:

$$pV_s = -\frac{(T_g + K_s K_g)}{(T_s T_g)}V_s - \frac{K_s}{(T_s T_g)}V_f - \frac{K_s K_g}{(T_s T_g)}V_{err} \quad \text{(E.15)}$$

If the stabilizing feedback gain K_s is set to zero then the AVR reduces to a simple first order lag. If the open-circuit machine is also considered as a single lag stage then a second order closed loop system is formed. The resulting closed loop gain is of the form:

$$G_c(s) = \frac{K_c\omega_n^2}{s^2 + 2A_g\omega_n + \omega_n^2} \quad \text{(E.16)}$$

where K_c is the closed loop DC gain, ω_n is the natural frequency (rad/s) and A_g is the damping factor.

Since one per unit terminal voltage on open circuit is generated for one per unit field voltage, the machine has unity gain. Given the machine time constant, T'_{do}, and the AVR gain, K_g, the AVR time constant, T_g, can be determined for a specified damping factor, A_g.

The governor model assumes an infinite steam source at the high pressure cylinder

governor valve. The valve is controlled by machine speed and is followed by two time constants representing steam flows through the high pressure pipe work and the reheater. The following equations are yielded:

$$pV_{pos} = \frac{-1}{T_a}V_{pos} + \frac{(T_{mo} - K_t\omega)}{T_a} \quad \text{(E.17)}$$

$$pT_{m1} = \frac{1}{T_b}V_{pos} - \frac{1}{T_b}T_{m1} \quad \text{(E.18)}$$

$$pT_{m2} = \frac{1}{T_c}T_{m1} - \frac{1}{T_c}T_{m2} \quad \text{(E.19)}$$

Magnetic Saturation

So far, the dynamic and algebraic equations are for idealized synchronous machines and magnetic saturation is neglected. However, the importance of including magnetic saturation effects in response calculations had been demonstrated by the CEGB [118–120].

For a full representation of magnetic saturation, a step by step finite element analysis of the flux paths of every machine modeled is too expensive. Alternatively, it is possible to simplify the representation by using saturation factors obtained from the machine open circuit terminal voltage / field current curve. However, the non-linearities introduced during saturation makes this approach unsuitable.

A more practical approach is to assume that both mutual and leakage path of the rotor circuits are saturated equally [121]. The saturated subtransient reactances then vary slightly with saturation factor and it is justifiable to replace them with the unsaturated values. The differential equations can be rearranged into

a pseudo-linear form which is suitable for direct solution by introducing new non-integrable variables. The saturation factor is expressed as:

$$K = 1 + \alpha \tag{E.20}$$

and the non-integrable variables are:

$$pE''_d = \left((X_q - X''_q) \cdot I_q - E''_d - aE''_d\right) / T''_{qo} \tag{E.21}$$

$$pE'_q = \left(V_f - (X_d - X'_d) \cdot I_d - E'_q - aE'_q\right) / T'_{do} \tag{E.22}$$

$$pE''_q = \left(E'_q + aE'_q - (X'_d - X''_d) \cdot I_d - E''_q - aE''_q\right) / T''_{do} \tag{E.23}$$

To maintain the stability of the implicit integration technique, it is important to make the non-integrable variables as independent from the integrable variables as possible. For this reason I_d and I_q are replaced with V_d and V_q which are less strongly coupled to E''_d and E''_q.

Solution Techniques

The power system models used in the Simulator are described above and the resulting equations can be arranged in the following pseudo-linear state space form:

$$py = \mathbf{A} \cdot y + \mathbf{B} \cdot u(y, x) \tag{E.24}$$

where u is a vector of non-integrable variables, which are some non-linear function of y and x, such that $\mathbf{A}$ and $\mathbf{B}$ are constant matrices. As a direct solution for the integrable variables is possible, this form is particularly convenient for fast execution using implicit integration methods.

The simulator uses a partitioned method whereby the non-integrable variables x are extrapolated. The integrable variables y_{k+1} are obtained using an algebraic integration method. A better estimate of x_{k+1} is then computed from equation E.24 and the procedure iterated until x and y converge.

The implicit trapezoidal integration method was chosen for its stability properties which allow fairly large integration steps to be taken while solution reliability is ensured. The pseudo-linear form of the dynamic equations can be used to give constant integration matrices for a given system condition enabling fast calculation.

Extrapolation Functions

The non-integrable variables are extrapolated using a variable order function. The strategy adopted is described below:

1. Immediately following a network or local machine parameter discontinuity, a zero order function is used:

$$a_{k+1} = a_k \tag{E.25}$$

2. One step after a discontinuity, a first order function is used:

$$a_{k+1} = 2a_k - a_{k-1} \tag{E.26}$$

3. Two steps after a discontinuity and during normal running, a second order function is used:

$$a_{k+1} = 3a_k - 3a_{k-1} + a_{k-2} \tag{E.27}$$

The machine rotor angle is also extrapolated. When referred back to the network reference frame, the extrapolated busbar voltage can then have an appropriate phase shift.

Integration Algorithm

The implicit trapezoidal difference equation is:

$$y_{k+1} = y_k + \frac{h}{2}\left[\mathcal{F}(y_k) + \mathcal{F}(y_{k+1})\right] \tag{E.28}$$

When applied to the pseudo-linear matrix form of equation E.24 this yields:

$$y_{k+1} = y_k + \frac{h}{2}\left[\mathbf{A}y_k + \mathbf{B}u_k + \mathbf{A}y_{k+1} + \mathbf{B}u_{k+1}\right] \tag{E.29}$$

and hence:

$$y_{k+1} = \left[\mathbf{I} - \frac{h}{2}\mathbf{A}\right]^{-1} \cdot \left\{\left[\mathbf{I} + \frac{h}{2}\mathbf{A}\right] y_k + \frac{h}{2}\mathbf{B}(u_k + u_{k+1})\right\} \tag{E.30}$$

where $\mathbf{I}$ is the identity matrix.

Both $\mathbf{A}$ and $\mathbf{B}$ are sparse matrices and hence the solution of this set of linear equations can best be solved by direct Gaussian elimination rather than multiplying by the non-sparse inverse matrix. Gaussian elimination is equivalent to forward and backward substitution using triangular matrices.

Appendix F

Assembler listing of the keyboard and mouse data capturing routines

```
*       This will reside in EPROMS to set
*             1.    VSC
*             2.    MOUSE
*             3.    KEYBOARD
*       And then loop and read mouse whenever it is moved.
*
*       Also checks to see if the co-processor wants the 68070 to
*       carry out 16-bit read/write operations for it.
*
*       LAST UPDATED... 25-oct-89 RWD
*       KEYBOARD INTERRUPT ROUTINE ADDED ... 25-Jan-90 F.NG
*
*       The keyboard generates INT1 when a key is pressed and VSC
*       resets INT1 when reading data from LS322 (serial to parallel).
*       INT1 generates a level 4 interrupt to VSC which looks at INTVEC
*       and multiply it by 4 to get instruction which is a jump to
*       routine INTKEY. There the key data is read and bit 7 of LIR is
*       reset to enable further interrupt from keyboard.
*

PICR2   EQU     $80004027       ; peripheral interrupt control reg.
EPROM   EQU     $180008         ; EPROM ENTRY LOCATION
BERRVEC EQU     $8              ; BUS ERROR VECTOR LOCATION
LOGICTR EQU     $140000         ; LOCATION IN THIRD RAM BLOCK TO TRIGGER
*                               ; THE LOGIC ANALYSER
```

```
*
ACIA    EQU     $80002010       ; acia base addr.
UMR     EQU     $1              ; mode reg
USR     EQU     $3              ; status reg
UCS     EQU     $5              ; clock select reg
UCR     EQU     $7              ; command reg
THR     EQU     $9              ; transmit hold reg
RHR     EQU     $B              ; reciever hold reg
*
XLOC    EQU     $7FFF0          ; shared memory mouse x position
YLOC    EQU     $7FFF2          ; shared memory mouse y position
BUTTON  EQU     $7FFF4          ; shared memory mouse button information
MOFLAG  EQU     $7FFF8          ; shared memory mouse semaphore flag.
ACFLAG  EQU     $7FFF9          ; shared memory access semaphore flag.
CODATA  EQU     $7FFFA          ; data transfer location.
COADD   EQU     $7FFFC          ; access address pointer.
STACK   EQU     $7F800          ; 2000 byte for stack
*
FLAGVAL EQU     $FF             ; flag value for T800 ; means values
*                               ; have been updated
*
INTSON  EQU     $2000           ; enable interrupt -- > 0
INTSOFF EQU     $2700           ; 7 means interrupt level under 7 is
*                               ; disabled, i.e. no interrupt at all
*
*
* The following is specific to the keyboard interrupt routine
*
*
INTVEC1  EQU    $70             ; level 4 autovector for INT1 interrupt
INTVEC2  EQU    $F0             ; level 4 autovector for INT1 interrupt
KEYBOARD EQU    $1FFC01         ; keyboard access address pointer
KEYDATA  EQU    $7FFE2          ; keyboard data transfer location
KEYFLAG  EQU    $7FFE4          ; tells transputer if a key is pressed
IENABLE  EQU    $C0             ; tells LIR that we are using level 4
IDISABLE EQU    $8F             ; tells LIR that we are not using ints
IACK     EQU    $80             ; tells LIR to loose the interrupt
LIR      EQU    $80001001       ; register holding interrupt level

         RORG   $0
*
BEGIN:   EQU    *
*
         DC.L   STACK,EPROM     ; RESET VECTOR & STACK POINTER
*
START :  MOVE.W  #INTSOFF,SR    ; MASK OFF ALL INTERRUPTS !!!!!
```

```
         LEA.L   START,A0       ; GET THE ADDRESS OF THE START
         LEA.L   BERRVEC,A1     ; SET START OF VECTOR AREA
         MOVE.W  #$100,D0       ; SET UP A LOOP COUNTER
LP1:     MOVE.L  A0,(A1)+       ; POINT BUS ERROR VECTOR TO RE-START
         DBRA    D0,LP1         ; LOOP UNTIL $400 BYTES COPIED
*
*     KEYBOARD SETUP
*
        LEA.L   INTKEY,A0       ; address of keyborad interrupt routine
        MOVE.L  A0,INTVEC1      ; save pointer in the level 4 vector
        MOVE.L  A0,INTVEC2      ; save in 'on chip' vector, just incase
        ANDI.B  #IDISABLE,LIR   ; remove any pending interrupt
        MOVE.B  KEYBOARD,D0     ; clear int from keyboard after reset
        ANDI.B  #IDISABLE,LIR   ; remove any pending interrupt

*
*     VSC SETUP
*
        LEA.L   $1FFFE0,A0      ; point at the VSC registers.
        MOVE.W  #$168,0(A0)     ; set up a the csr
*
*  This sets:-
*
*  1. FAST PAGE DRAM
*  2. 133ns EPROM DTACK
*
* the next bit waits until the display has just entered frame fly-back.
*
VSC1:   BTST    #7,1(A0)        ; check the status register
        BNE.S   VSC1            ; wait for it
VSC2:   BTST    #7,1(A0)        ; check it again, now it is low
        BEQ.S   VSC2            ; and wait on it to go high again
*
        MOVE.W  #$DF00,2(A0)    ; setup DCR1 in VSC
*
*       THIS GIVES...
*
* 28 MHz clock
* display enabled
* 65 Hz scan
* interlaced
* normal frequency
* non-full screen
* logical screen
* 4 bits per pixel
* no frame grab
```

```
* no ICA, no DCA
* start address = 2
*
        MOVE.W  #$400,4(A0)     ; lower 16 address bits in VSR = $400
        CLR.B   7(A0)           ; border=black.
*
        RESET                   ; Obvious Really
*
        MOVEA.L #XLOC,A0        ; Last four long words to clear,
        MOVE.L  #$3,D0          ; Set up a loop counter
VSC4:   CLR.L   (A0)+           ;
        DBRA    D0,VSC4         ;
*
*       To set up UART
*       MOUSE defaults to 1200 baud
*
        LEA.L   ACIA,A0         ; shift it into an address register
        MOVE.B  #0,PICR2        ; initialise the picr2 register
        MOVE.B  #$05,UCR(A0)    ; enable the receiver and transmitter
ACIA1:  BTST    #3,USR(A0)      ; see if the transmit shift reg is empty
        BEQ.S   ACIA1           ; wait until it is empty
        MOVE.B  #$0A,UCR(A0)    ; disable the receiver and transmitter
        MOVE.B  #$13,UMR(A0)    ; 8 bits no parity 2 stop bit & CTS
        MOVE.B  #$33,UCS(A0)    ; set recieve and transmit to 1200 baud
        MOVE.B  #$2A,UCR(A0)    ; reset the reciever, DISABLE TX RX
        MOVE.B  #$3A,UCR(A0)    ; reset the transmitter, DISABLE TX RX
        MOVE.B  #$4A,UCR(A0)    ; reset the error status, DISABLE TX RX
        MOVE.B  #$05,UCR(A0)    ; now enable the reciever & transmitter
*
*        The Internal Loop.
*        The higher and lower order bytes are swaped
*        to make it comptable with T800.
*
         CLR.B    MOFLAG        ; initialise the mouse flag
         CLR.B    ACFLAG        ; initialise the access flag
         CLR.B    KEYFLAG       ; initialise the keyboard flag
         MOVE.W   #INTSON,SR    ; turn on all interrupts
         ORI.B    #IENABLE,LIR  ; set INT1 to level 4
ENTERLOOP:
*
ICH1:
         MOVE.B   ACFLAG,D5     ; TEST TO SEE IF AN ACCESS IS REQUIRED
         BEQ.S    NOACC1        ; IF =0 THEN NO ACCESS REQUEST
ADD1:    MOVE.L   COADD,D6      ; GET THE ACCESS ADDRESS ***
OK12:    ROL.W    #8,D6         ; SHUFFLE
         SWAP     D6            ; IT
```

```
          ROL.W    #8,D6          ; ABOUT TO CORRECT FORM
          MOVE.L   D6,A1          ; AND PUT IT IN AN ADDRESS REGISTER.
          CMP.B    #1,D5          ; SEE IF IT IS A READ
          BEQ.S    COREAD1        ; IF READ THEN READ DATA
          MOVE.W   CODATA,0(A1)   ; ELSE, WRITE THE DATA
          BRA.S    COEND1         ; GO TO END
COREAD1:  MOVE.W   0(A1),CODATA   ; STORE THE DATA TO BE READ
COEND1:   CLR.B    ACFLAG         ; RESET THE FLAG TO INDICATE COMPLETION
NOACC1:
*
          BTST     #0,USR(A0)     ; GOT A CHAR YET
          BEQ.S    ICH1           ; IF NOT THEN WAIT
          MOVE.B   USR(A0),D0     ; GET THE UART STATUS
          ANDI.B   #$F0,D0        ; STRIP FOR ERROR BITS
          BNE      UERR           ; IF NOT OK THEN DUMP AND RESET
          MOVE.B   RHR(A0),D0     ; ELSE, GET THE DATA TO D0
*
          MOVE.B   D0,D1          ; MAKE A COPY OF IT
          ANDI.B   #$F8,D1        ; STRIP OUT THE SWITCH BITS
          CMPI.B   #$80,D1        ; CHECK TO SEE IF IT IS A SWITCH BYTE
          BNE      ENTERLOOP      ; IF NOT THEN DUMP IT
          MOVE.B   D0,BUTTON      ; byte 1 = BUTTON
*
ICH2:
          MOVE.B   ACFLAG,D5      ; TEST TO SEE IF AN ACCESS IS REQUIRED
          BEQ.S    NOACC2         ; IF =0 THEN NO ACCESS REQUEST
ADD2:     MOVE.L   COADD,D6       ; GET THE ACCESS ADDRESS ***
OK22:     ROL.W    #8,D6          ; SHUFFLE
          SWAP     D6             ; IT
          ROL.W    #8,D6          ; ABOUT TO CORRECT FORM
          MOVE.L   D6,A1          ; AND PUT IT IN AN ADDRESS REGISTER.
          CMP.B    #1,D5          ; SEE IF IT IS A READ
          BEQ.S    COREAD2        ; IF READ THEN READ DATA
          MOVE.W   CODATA,0(A1)   ; ELSE, WRITE THE DATA
          BRA.S    COEND2         ; GO TO END
COREAD2:  MOVE.W   0(A1),CODATA   ; STORE THE DATA TO BE READ
COEND2:   CLR.B    ACFLAG         ; RESET THE FLAG TO INDICATE COMPLETION
NOACC2:
*
          BTST     #0,USR(A0)     ; GOT A CHAR YET
          BEQ.S    ICH2           ; IF NOT THEN WAIT
          MOVE.B   USR(A0),D2     ; GET THE UART STATUS
          ANDI.B   #$F0,D2        ; STRIP FOR ERROR BITS
          BNE      UERR           ; IF NOT OK THEN DUMP AND RESET
          MOVE.B   RHR(A0),D2     ; ELSE, GET THE DATA TO D0
*
```

```
ICH3:
        MOVE.B   ACFLAG,D5     ; TEST TO SEE IF AN ACCESS IS REQUIRED
        BEQ.S    NOACC3        ; IF =0 THEN NO ACCESS REQUEST
ADD3:   MOVE.L   COADD,D6      ; GET THE ACCESS ADDRESS ***
OK32:   ROL.W    #8,D6         ; SHUFFLE
        SWAP     D6            ; IT
        ROL.W    #8,D6         ; ABOUT TO CORRECT FORM
        MOVE.L   D6,A1         ; AND PUT IT IN AN ADDRESS REGISTER.
        CMP.B    #1,D5         ; SEE IF IT IS A READ
        BEQ.S    COREAD3       ; IF READ THEN READ DATA
        MOVE.W   CODATA,0(A1)  ; ELSE, WRITE THE DATA
        BRA.S    COEND3        ; GO TO END
COREAD3: MOVE.W  0(A1),CODATA  ; STORE THE DATA TO BE READ
COEND3:  CLR.B   ACFLAG        ; RESET THE FLAG TO INDICATE COMPLETION
NOACC3:
*
        BTST     #0,USR(A0)    ; GOT A CHAR YET
        BEQ.S    ICH3          ; IF NOT THEN WAIT
        MOVE.B   USR(A0),D3    ; GET THE UART STATUS
        ANDI.B   #$F0,D3       ; STRIP FOR ERROR BITS
        BNE      UERR          ; IF NOT OK THEN DUMP AND RESET
        MOVE.B   RHR(A0),D3    ; ELSE, GET THE DATA TO D0
*
ICH4:
        MOVE.B   ACFLAG,D5     ; TEST TO SEE IF AN ACCESS IS REQUIRED
        BEQ.S    NOACC4        ; IF =0 THEN NO ACCESS REQUEST
ADD4:   MOVE.L   COADD,D6      ; GET THE ACCESS ADDRESS ***
OK42:   ROL.W    #8,D6         ; SHUFFLE
        SWAP     D6            ; IT
        ROL.W    #8,D6         ; ABOUT TO CORRECT FORM
        MOVE.L   D6,A1         ; AND PUT IT IN AN ADDRESS REGISTER.
        CMP.B    #1,D5         ; SEE IF IT IS A READ
        BEQ.S    COREAD4       ; IF READ THEN READ DATA
        MOVE.W   CODATA,0(A1)  ; ELSE, WRITE THE DATA
        BRA.S    COEND4        ; GO TO END
COREAD4: MOVE.W  0(A1),CODATA  ; STORE THE DATA TO BE READ
COEND4:  CLR.B   ACFLAG        ; RESET THE FLAG TO INDICATE COMPLETION
NOACC4:
*
        BTST     #0,USR(A0)    ; GOT A CHAR YET
        BEQ.S    ICH4          ; IF NOT THEN WAIT
        MOVE.B   USR(A0),D0    ; GET THE UART STATUS
        ANDI.B   #$F0,D0       ; STRIP FOR ERROR BITS
        BNE      UERR          ; IF NOT OK THEN DUMP AND RESET
        MOVE.B   RHR(A0),D0    ; ELSE, GET THE DATA TO D0
*
```

```
        ADD.B   D0,D2          ; ADD TO FIRST X INCREMENT
        EXT.W   D2             ; sign extend it
        MOVE.W  D2,XLOC        ; copy it to the required location.
*
ICH5:
        MOVE.B  ACFLAG,D5      ; TEST TO SEE IF AN ACCESS IS REQUIRED
        BEQ.S   NOACC5         ; IF =0 THEN NO ACCESS REQUEST
ADD5:   MOVE.L  COADD,D6       ; GET THE ACCESS ADDRESS ***
OK52:   ROL.W   #8,D6          ; SHUFFLE
        SWAP    D6             ; IT
        ROL.W   #8,D6          ; ABOUT TO CORRECT FORM
        MOVE.L  D6,A1          ; AND PUT IT IN AN ADDRESS REGISTER.
        CMP.B   #1,D5          ; SEE IF IT IS A READ
        BEQ.S   COREAD5        ; IF READ THEN READ DATA
        MOVE.W  CODATA,0(A1)   ; ELSE, WRITE THE DATA
        BRA.S   COEND5         ; GO TO END
COREAD5: MOVE.W 0(A1),CODATA   ; STORE THE DATA TO BE READ
COEND5: CLR.B   ACFLAG         ; RESET THE FLAG TO INDICATE COMPLETION
NOACC5:
*
        BTST    #0,USR(A0)     ; GOT A CHAR YET
        BEQ.S   ICH5           ; IF NOT THEN WAIT
        MOVE.B  USR(A0),D0     ; GET THE UART STATUS
        ANDI.B  #$F0,D0        ; STRIP FOR ERROR BITS
        BNE.S   UERR           ; IF NOT OK THEN DUMP AND RESET
        MOVE.B  RHR(A0),D0     ; ELSE, GET THE DATA TO D0
*
        ADD.B   D0,D3          ; ADD TO FIRST Y INCREMENT
        EXT.W   D3             ; sign extend it
        MOVE.W  D3,YLOC        ; copy it to the required location.
*
        MOVE.B  #FLAGVAL,MOFLAG  ; set up flag and
        BRA     ENTERLOOP      ; start again
*
UERR:   MOVE.B  #$06,UCR(A0)   ; RESET THE ERROR BITS, DISABLE RECEIVER
        MOVE.B  #$05,UCR(A0)   ; RE-ENABLE THE RECEIVER
        BRA     ENTERLOOP      ; START AGAIN
*

*
* KEYBOARD INTERRUPT ROUTINE FOLLOWS :-
*

INTKEY: MOVE.B KEYBOARD,KEYDATA ; read  data in
        MOVE.B #FLAGVAL,KEYFLAG ; set keyboard flag to indicate new data
        ANDI.B #IDISABLE,LIR    ; remove any pending interrupts
```

```
        ORI.B   #IACK,LIR       ; acknowledge any interrupt to make sure
        ORI.B   #IENABLE,LIR    ; set interrupts to level 4 i.e. ON
        RTE                     ; return

FIN:            END
```

Appendix G

Circuit Diagram

Sheet 1 and 2

Graphics Board for the Five Transputer Rack.

Appendix H

Screen Dumps

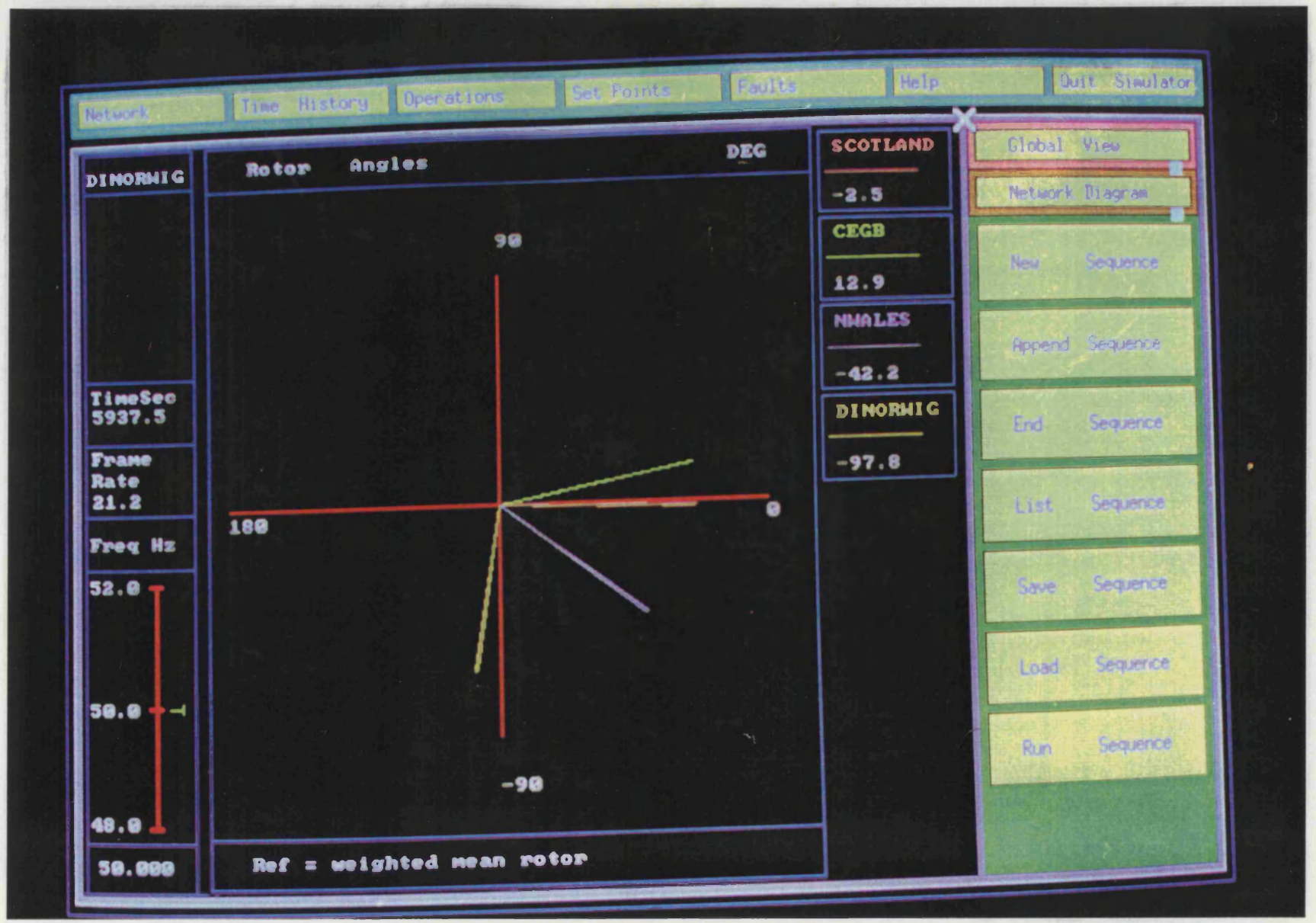


Fig. H.1 A detailed vector diagram of four machines

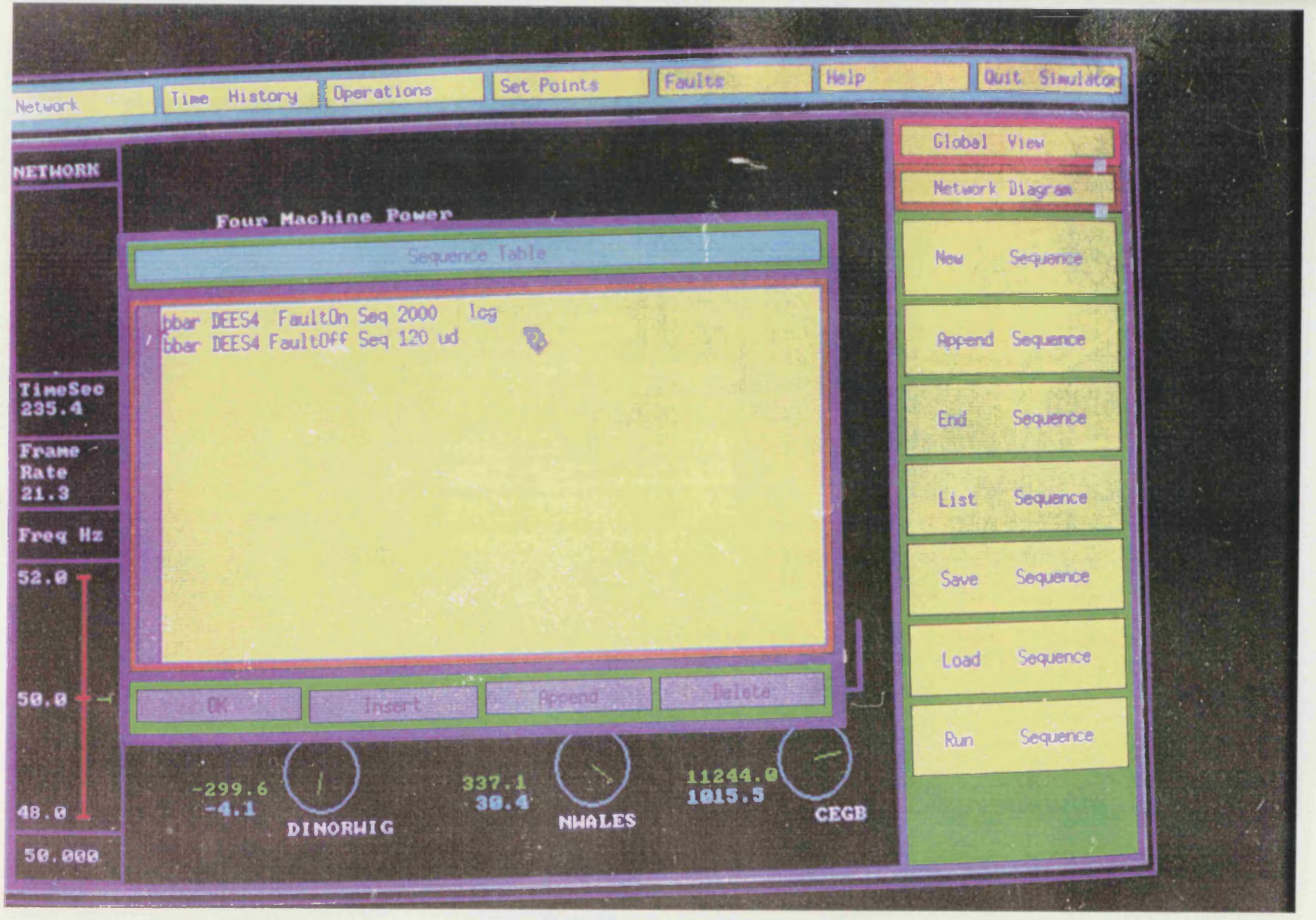


Fig. H.2 Table displaying the content of a selected file

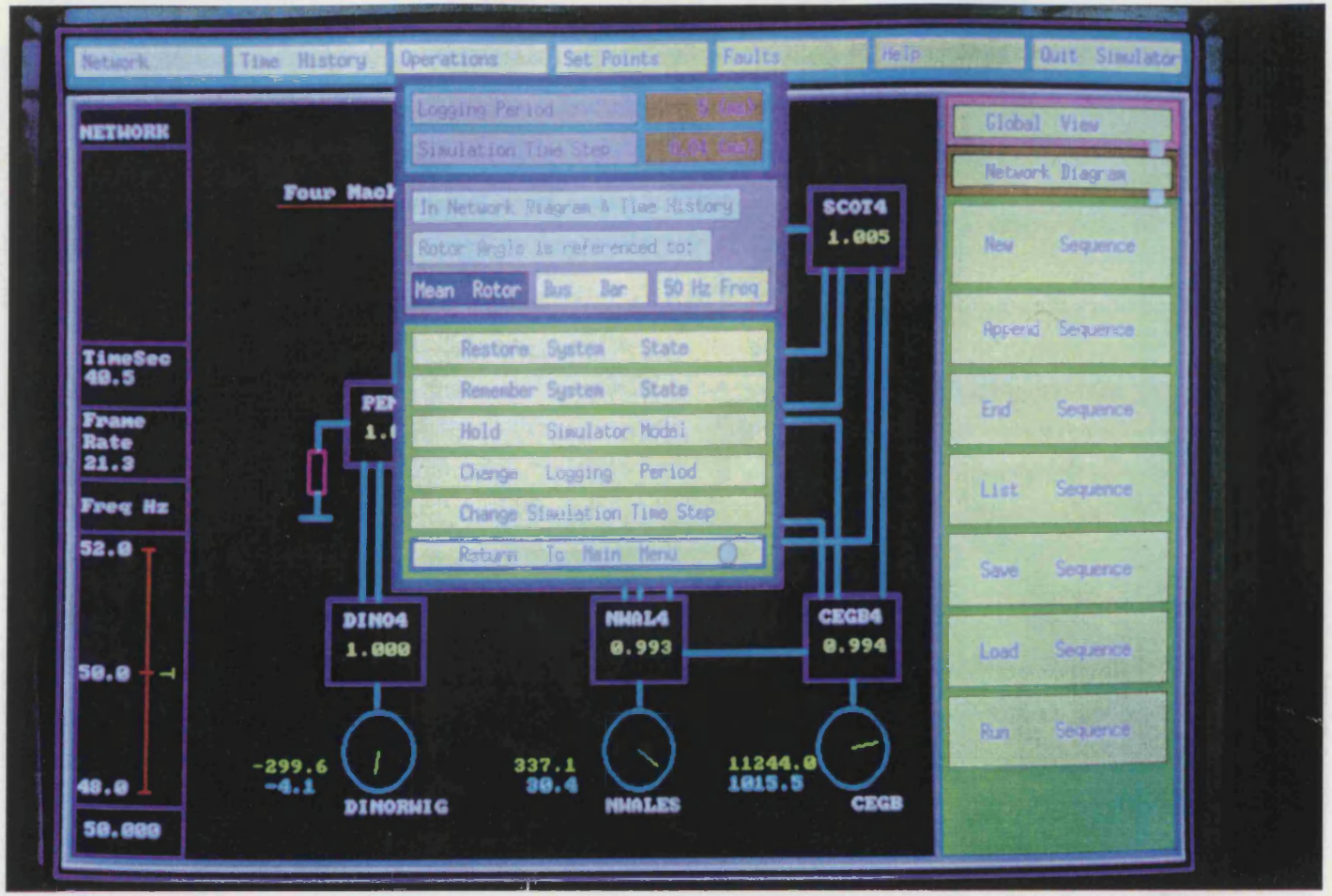


Fig. H.3 The operation menu

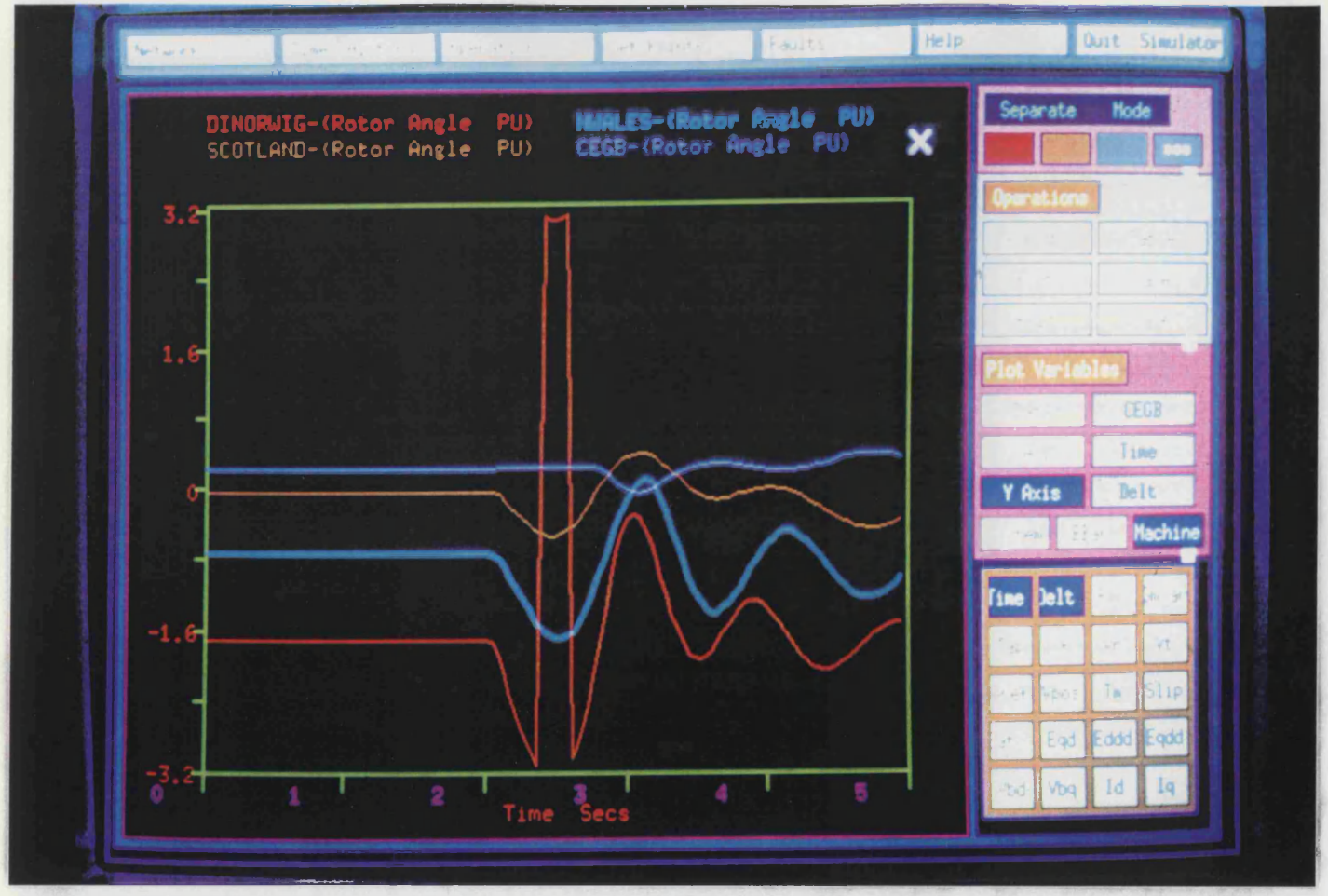


Fig. H.4 Four superimposed graphs

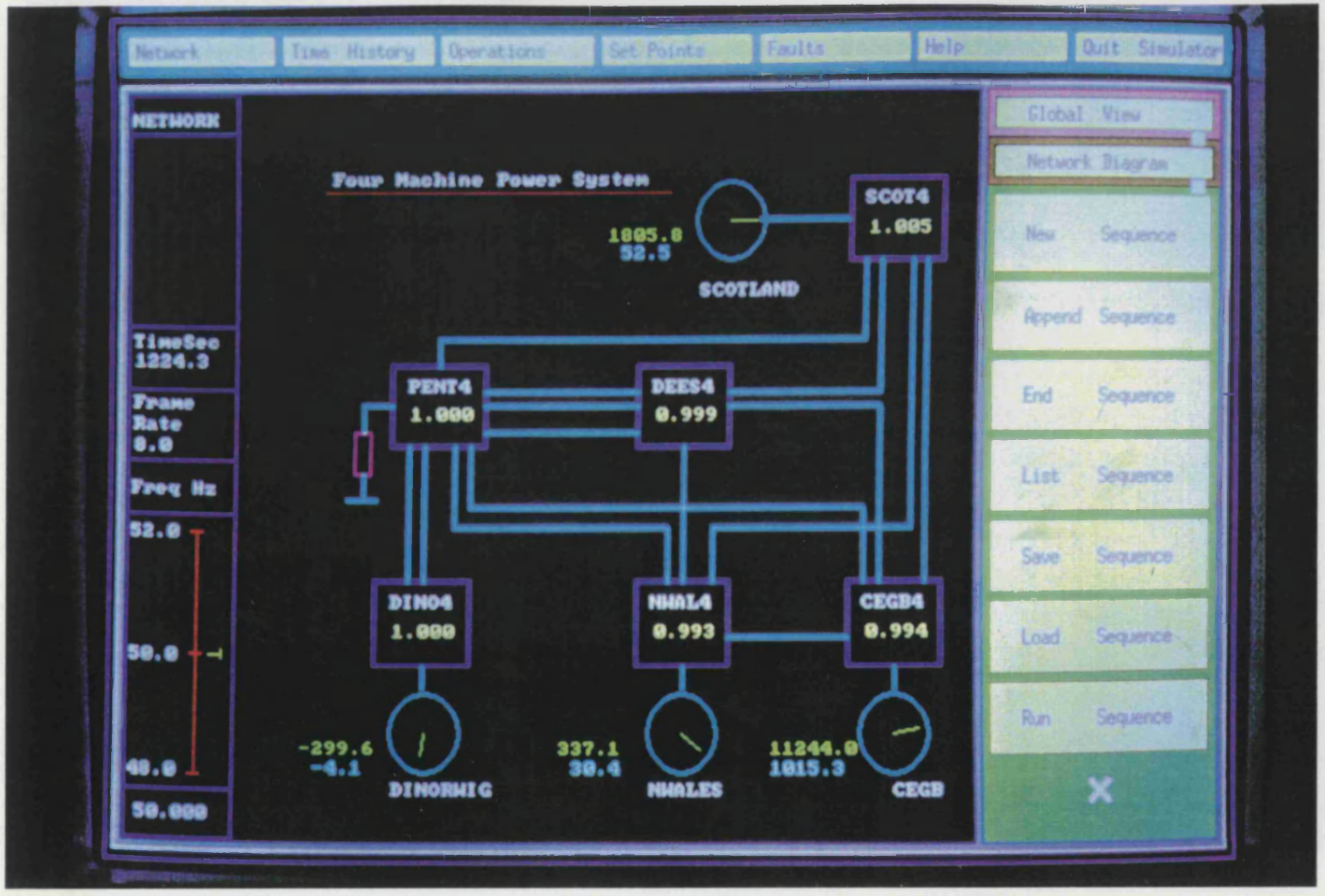


Fig. H.5 One-line diagram of the four machine study

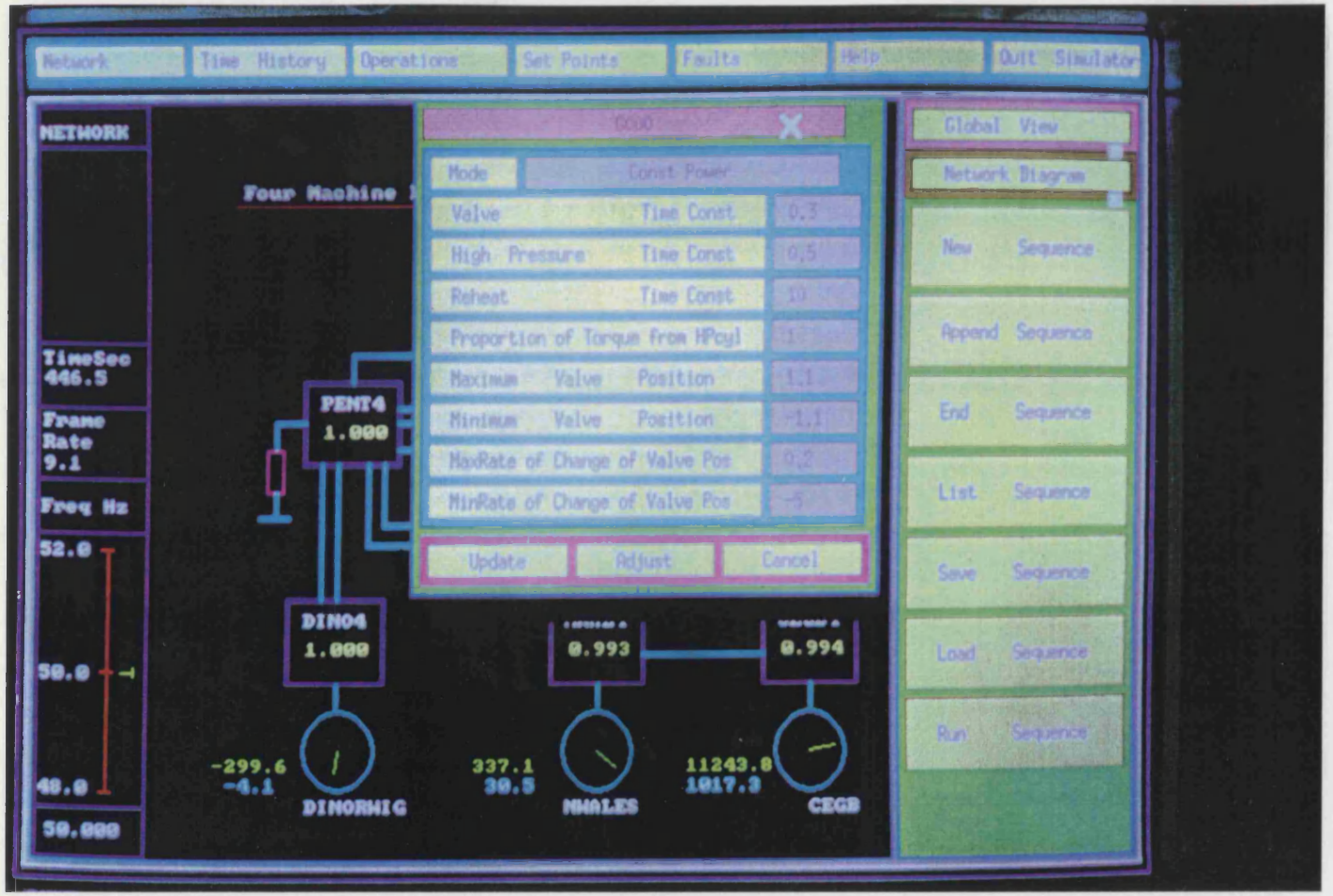


Fig. H.6 Menu of set points of a governor

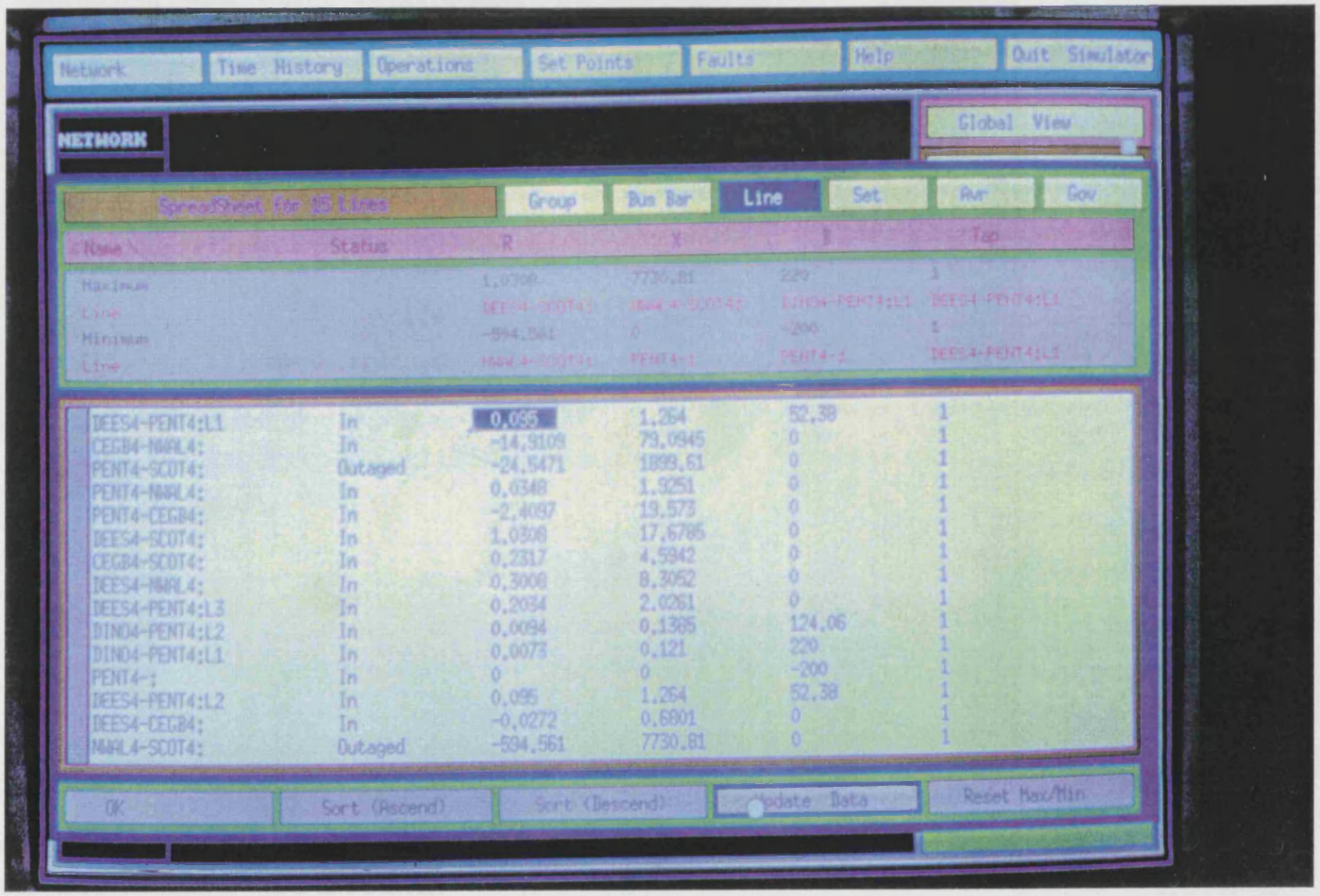


Fig. H.7 Spreadsheet of all the available lines

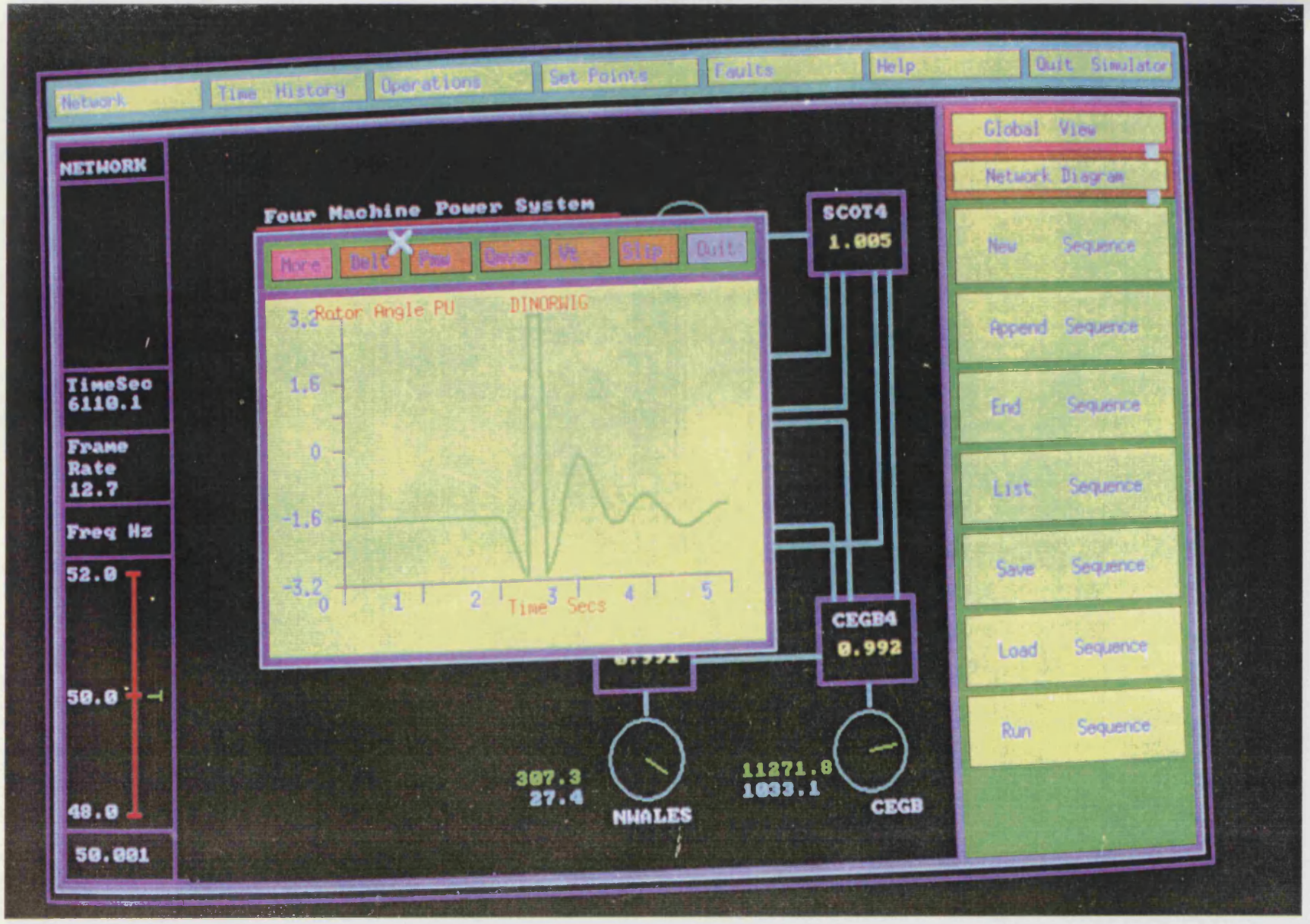


Fig. H.8 Time history plot of a machine

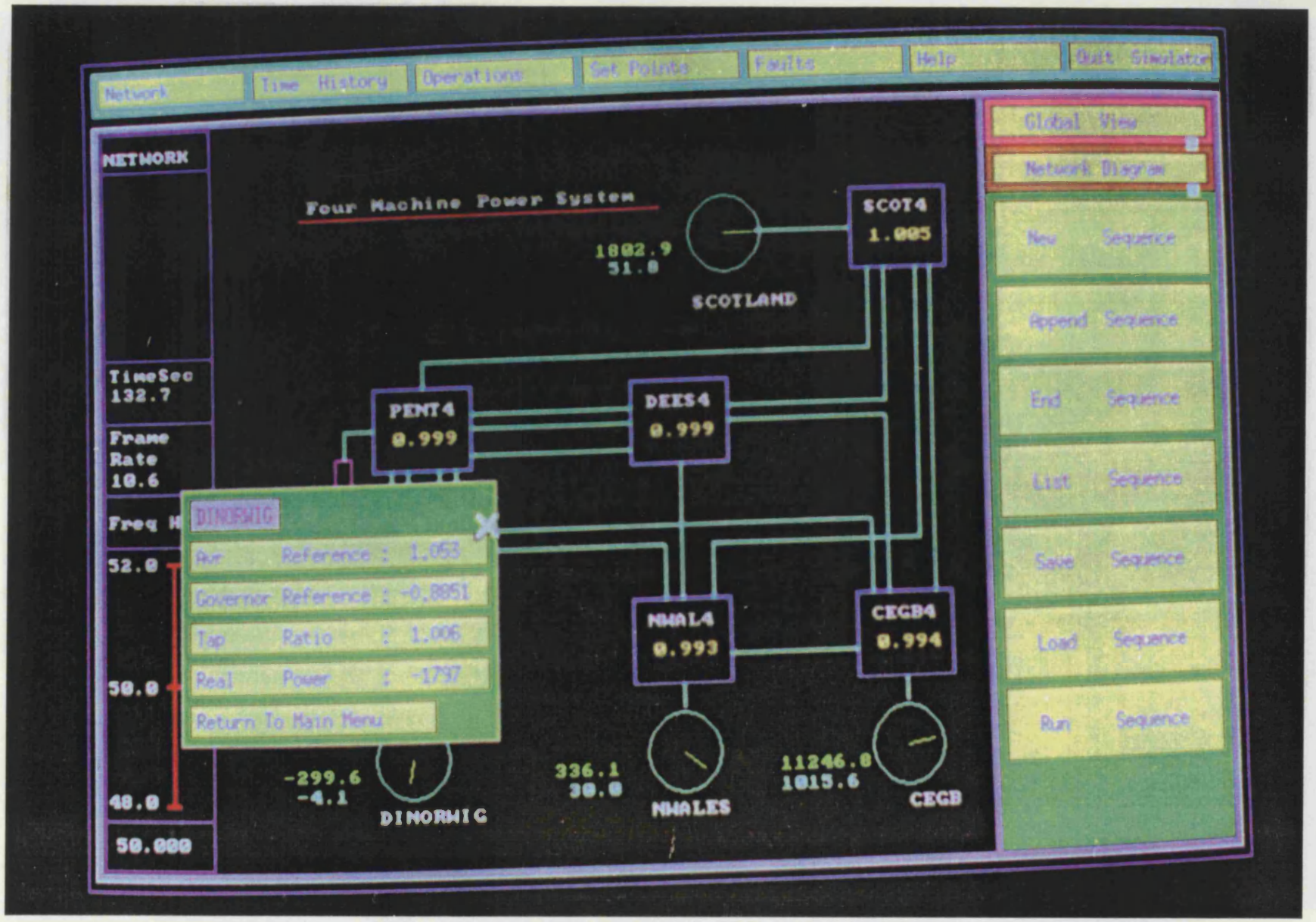


Fig. H.9 Menu of the set points of a machine

Fig. H.10 Four separately displayed graphs

Fig. H.11 Running graph of a machine

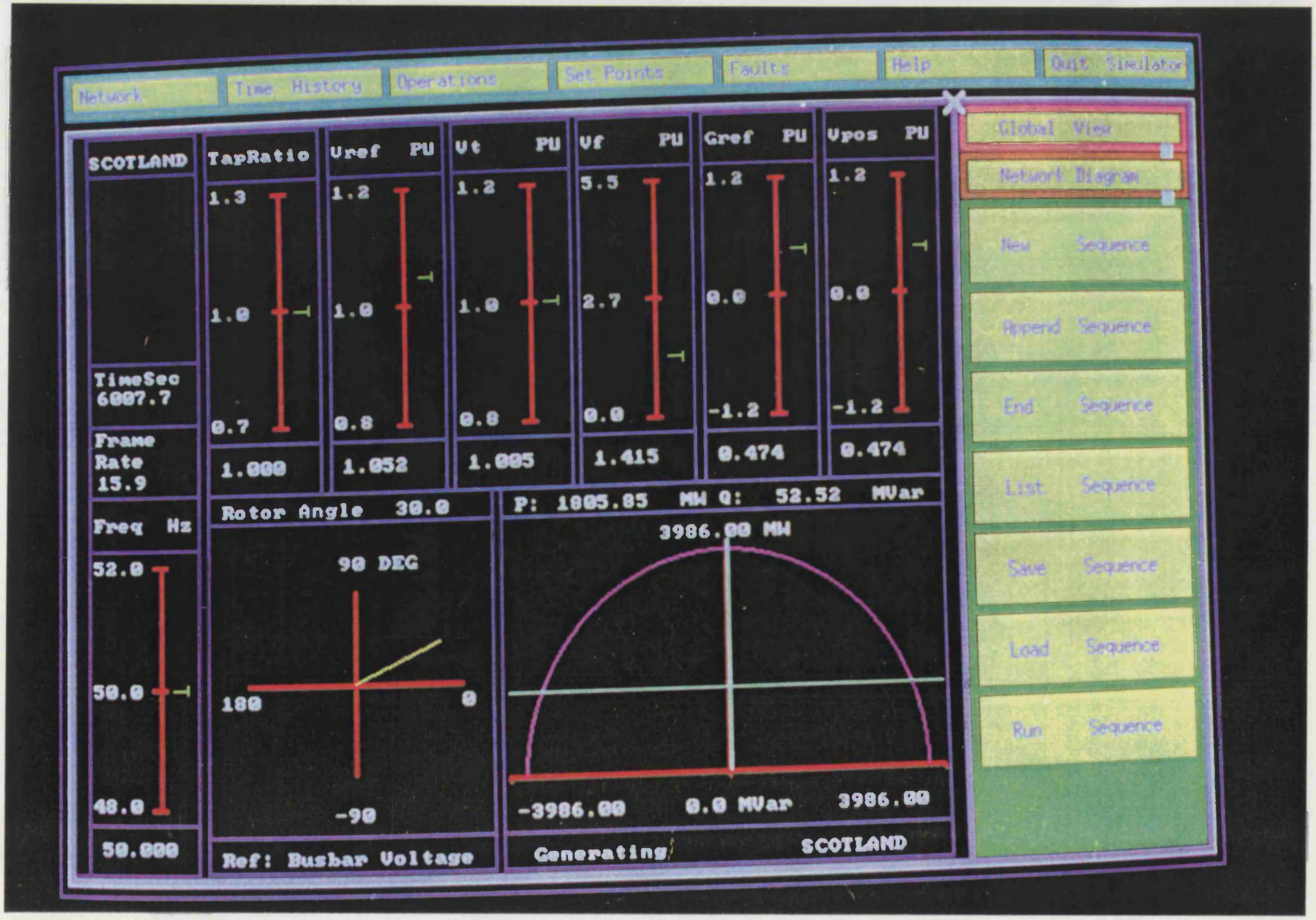


Fig. H.12 A detailed machine diagram

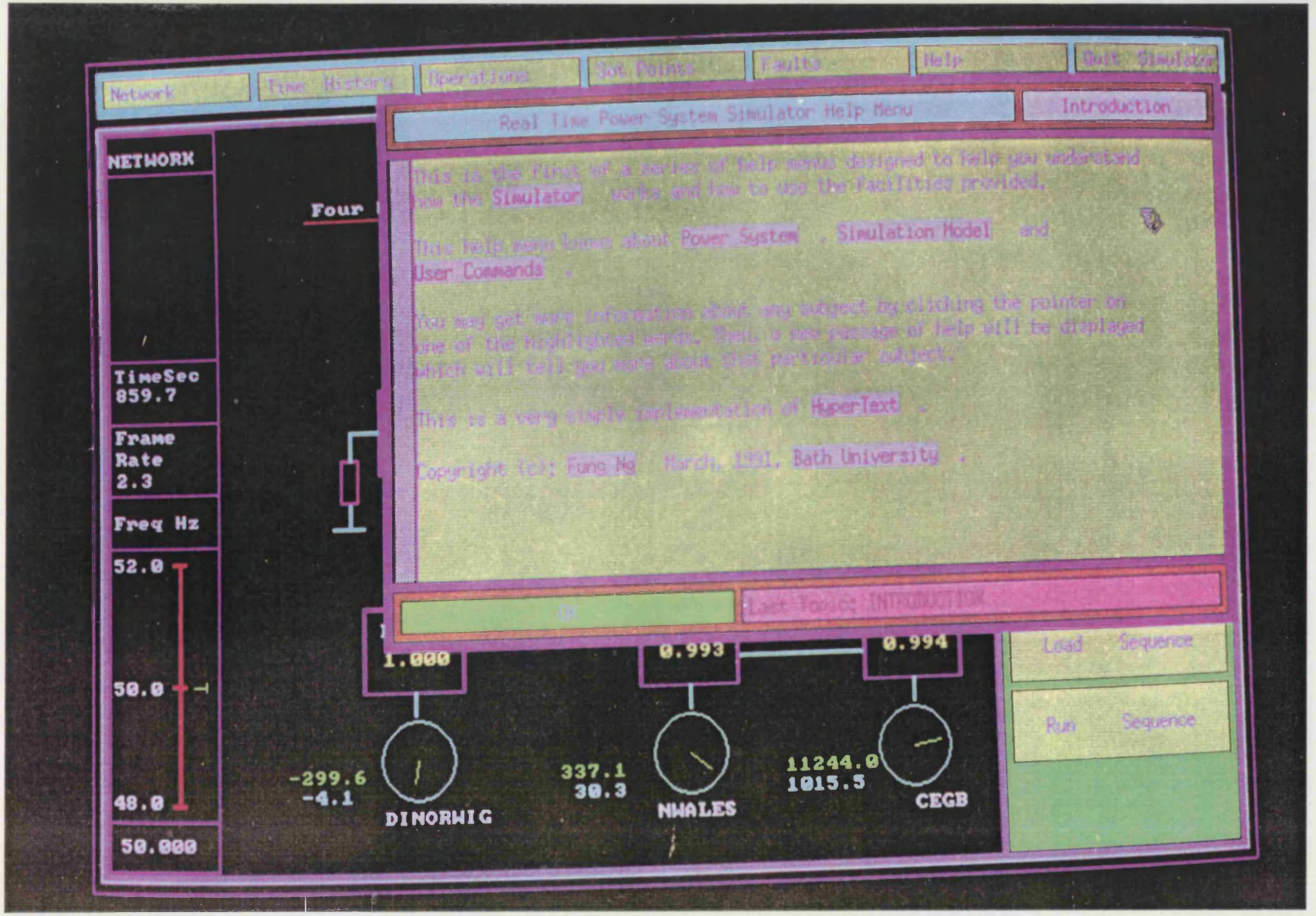
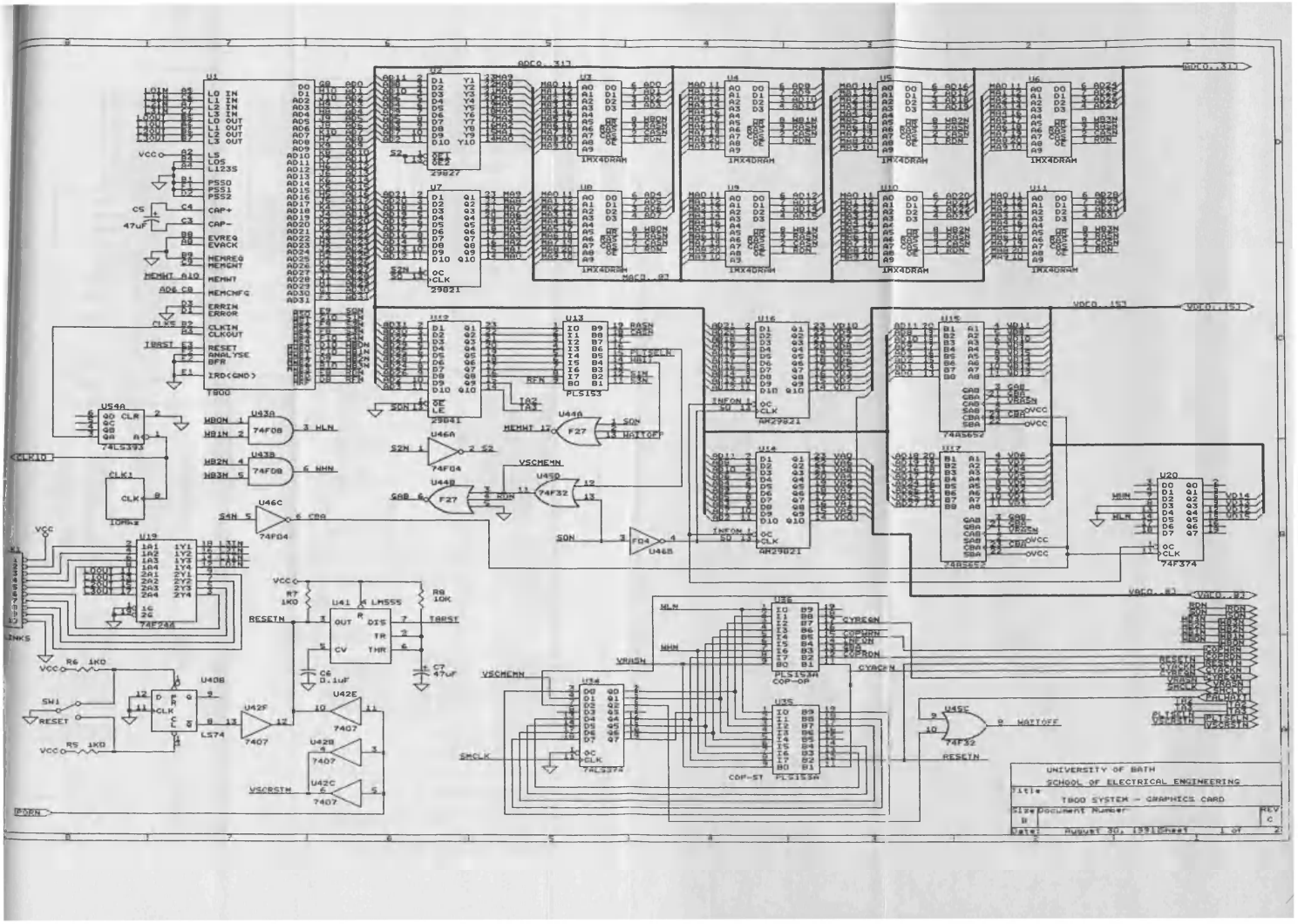
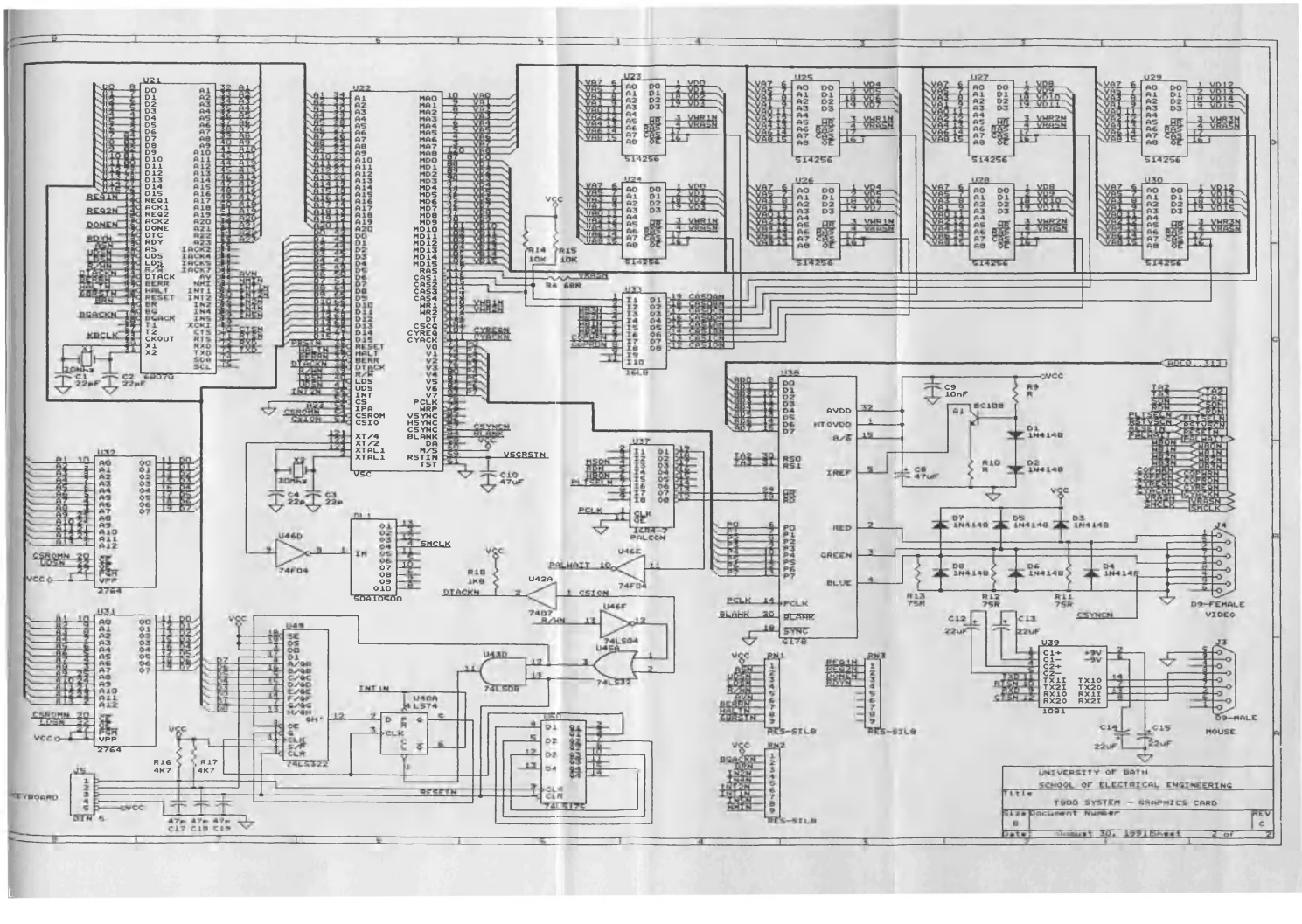


Fig. H.13 Hypertext help menu


UNIVERSITY OF BATH
SCHOOL OF ELECTRICAL ENGINEERING
Title
T800 SYSTEM - GRAPHICS CARD
Size
B
Document Number
REV
C
Date
August 30, 1991
Sheet
1 of
2


UNIVERSITY OF BATH
SCHOOL OF ELECTRICAL ENGINEERING
Title
T800 SYSTEM - GRAPHICS CARD
Size
Document Number
REV
C
Date
August 30, 1991
Sheet
2 of
2
D9-FEMALE
VIDEO
D9-MALE
MOUSE
KEYBOARD
CSYNCN
VCC
RESETN